Attention mechanism for sentiment analysis

In this lab we use part of the 'Amazon_Unlocked_Mobile.csv' dataset published by Kaggle. The dataset contain the following information:

- Product Name
- Brand Name
- Price
- Rating
- Reviews
- Review Votes

We are mainly interested by the 'Reviews' (X) and by the 'Rating' (y)

The goal is to try to predict the 'Rating' after reading the 'Reviews'. I've prepared for you TRAIN and TEST set. The work to be done is as follows:

1. Feature extraction and baseline
    - read the dataset and understand it
    - put it in a format so that you can use `CountVectorizer` or `Tf-IDF` to extract the desired features
    - perform on the desired dates and preprocessing
    - use one of the classifiers you know to predict the polarity of different sentences
2. My first neural network
    - reuse the features already extracted
    - proposed a neural network built with Keras
3. Hyper-parameter fitting
    - for the base line: adjust min_df, max_df, ngram, max_features + model's hyper-parameter
    - for the neural network: adjust batch size, number of layers and number of neuron by layers, use earlystop
4. Word embedding
    - stage 1 build a network that uses Keras' embedding which is not language sensitive.
    - stage 2 build a network that simultaneously uses Keras' embedding and the features extracted in the first weeks.
    - stage 3 try to use an existing embedding (https://github.com/facebookresearch/MUSE)

WARNING: the dataset is voluminous, I can only encourage you to work first on a small part of it and only at the end, when the code is well debugged and that it is necessary to build the "final model", to use the whole dataset.

In [ ]:
```
"""
(Practical tip) Table of contents can be compiled directly in jupyter noteb
I set an exception: if the package is in your installation you can import i
then import it.
"""
try:
```

```
    from jyquickhelper import add_notebook_menu
except:
    !pip install jyquickhelper
    from jyquickhelper import add_notebook_menu

"""
Output Table of contents to navigate easily in the notebook.
For interested readers, the package also includes Ipython magic commands to
wherever you are in the notebook to look for cells faster
"""
add_notebook_menu()
```

Out[ ]: run previous cell, wait for 2 seconds

Imports

In [ ]:
```
import os

import keras_tuner as kt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.strings
import tensorflow_addons as tfa
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import (LSTM, Activation, AveragePooling1D,
                                     Bidirectional, Dense, Dot, Dropout,
                                     Embedding, Flatten, Input, Permute,
                                     RepeatVector, TextVectorization,
                                     TimeDistributed)
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.utils import plot_model
from tensorflow_addons.metrics import F1Score
```

Read the dataset

Could you find below a proposal. You can complete them.

In [ ]:
```
BASE_DATASET_PATH = "http://www.i3s.unice.fr/~riveill/dataset/Amazon_Unlock
TRAIN = pd.read_csv(f"{BASE_DATASET_PATH}train.csv.gz").fillna(value="")
VAL = pd.read_csv(f"{BASE_DATASET_PATH}val.csv.gz").fillna(value="")
TEST = pd.read_csv(f"{BASE_DATASET_PATH}test.csv.gz").fillna(value="")

# TRAIN = TRAIN[:2042]  # save training time

TRAIN.head()
```

Out[ ]:

	Product Name	Brand Name	Price	Rating	Reviews	Review Votes
0	Samsung Galaxy Note 4 N910C Unlocked Cellphone...	Samsung	449.99	4	I love it!!! I absolutely love it!! 👌👍	0.0
1	BLU Energy X Plus Smartphone - With 4000 mAh S...	BLU	139.0	5	I love the BLU phones! This is my second one t...	4.0
2	Apple iPhone 6 128GB Silver AT&T	Apple	599.95	5	Great phone	1.0
3	BLU Advance 4.0L Unlocked Smartphone -US GSM -...	BLU	51.99	4	Very happy with the performance. The apps work...	2.0
4	Huawei P8 Lite US Version- 5 Unlocked Android ...	Huawei	198.99	5	Easy to use great price	0.0

In [ ]:
```
""" Construct X_train and y_train """
X_train = np.array(TRAIN["Reviews"]).reshape(-1, 1)

ohe = OneHotEncoder(sparse=False, handle_unknown="ignore")
y_train = ohe.fit_transform(np.array(TRAIN["Rating"]).reshape(-1, 1))

X_train.shape, y_train.shape
```

Out[ ]:
```
((5000, 1), (5000, 5))
```

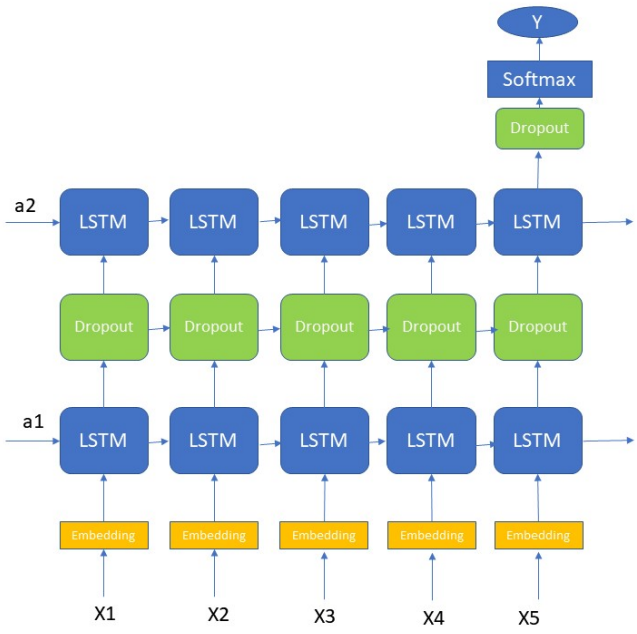
In [ ]:
```
""" Do the same for val """
X_val = np.array(VAL["Reviews"]).reshape(-1, 1)
y_val = ohe.transform(np.array(VAL["Rating"]).reshape(-1, 1))

""" Do the same for test """
X_test = np.array(TEST["Reviews"]).reshape(-1, 1)
y_test = ohe.transform(np.array(TEST["Rating"]).reshape(-1, 1))
```

In [ ]:
```
X_train.shape, X_test.shape, X_val.shape
```

Out[ ]:
```
((5000, 1), (1000, 1), (1000, 1))
```

Build an a neural network with vectorized embedding and RNN cells.

The task is to predict the sentiment according to the content of the review. We can treat this kind of task by a Many-to-one model.



Implement such a network with :

- a first layer of type LSTM
- a second layer of type LSTM, each cell of this layer will be fed by the corresponding output of the first layer (see figure above).

Study of the size of the reviews.

In [ ]:
```
import plotly.express as px

px.histogram(
    data_frame={"review_length": [len(review[0].split(" ")) for review in X
    cumulative=True,
    histnorm="probability density",
    title="Cumulative histogram of the review lengths (in number of words)"
)
```

We simply spitted the reviews by white space, which is a simple, basic approach, but yet informative. The plot above tells us that in order to get 95% of the reviews that are non-truncated, we could consider the `max_len` parameter to be 150.

In [ ]:
```
# Constants
nb_classes = y_train.shape[1]
vocab_size = 10 ** 4  # Maximum vocab size -- adjust with the size of the v
embedding_size = 20  # Embedding size (usually <= 300)
recurrent_size = 64  # Recurrent size
hidden_size = recurrent_size // 4  # Hidden layer
dropout_rate = 0.2  # Dropout rate for regularization (usually between 0.1
max_len = 150  # Sequence length to pad the outputs to (deduced from the le
learning_rate = 0.0075
```

Try to deal with emojis

Here, I try to make a callable that makes emojis behave as words, but it is not successful. I thought leaving traces of some work is better than showing nothing at all.\ I dove into the

tensorflow documentation to find what the `"strip_and_lower"` keyword actually did. I found that it uses `tf.strings.lower` and `tf.strings.regex_replace`, but I couldnt get it to split my emojis with a blank space.

In [ ]:
```
emoji = X_train[0,0][-1]
DEFAULT_STRIP_REGEX = r'[!"#$%&()\*\+,-\./:;<=>?@\[\\\]^_`{|}~\']'
DEFAULT_STRIP_REGEX
```

Out[ ]:
```
'[!"#$%&()\\*\\+,-\\./:;<=>?@\\[\\\\\\]^_`{|}~\\\']'
```

In [ ]:
```
X_train[0,0].encode("unicode_escape")
X_train_transf = tensorflow.strings.regex_replace(X_train, r"\\", "RRR")
X_train_transf = tensorflow.strings.lower(X_train_transf)
# X_train_transf = tensorflow.strings.regex_replace(X_train_transf, DEFAULT
X_train_transf[0,0].numpy()
```

```
2022-02-13 21:46:52.018822: I tensorflow/stream_executor/cuda/cuda_gpu_exec
utor.cc:939] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2022-02-13 21:46:52.062348: W tensorflow/stream_executor/platform/default/d
so_loader.cc:64] Could not load dynamic library 'libcudnn.so.8'; dlerror: l
ibcudnn.so.8: cannot open shared object file: No such file or directory
2022-02-13 21:46:52.062373: W tensorflow/core/common_runtime/gpu/gpu_devic
e.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing l
ibraries mentioned above are installed properly if you would like to use GP
U. Follow the guide at https://www.tensorflow.org/install/gpu for how to do
wnload and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-02-13 21:46:52.063190: I tensorflow/core/platform/cpu_feature_guard.c
c:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

Out[ ]:
```
b'i love it!!! i absolutely love it!! \xf0\x9f\x91\x8c\xf0\x9f\x91\x8d'
```

In [ ]:
```
print(r"".format(X_train[0,0][-1]))
```

In [ ]:
```
# Create the vectorized layer.
vectorize_layer = TextVectorization(
    max_tokens=vocab_size,
    standardize="lower_and_strip_punctuation",
    # it is possible to build your own function
    # to transform emoji into text
    # to transform foreign reviews in english one
    # etc.
    output_mode="int",
    output_sequence_length=max_len,
)
```

In [ ]:
```
# Fit vectorized layer on train
vectorize_layer.adapt(X_train)
```

In [ ]:
```
vectorize_layer(X_train[0])
```

Out[ ]:
```
<tf.Tensor: shape=(1, 150), dtype=int64, numpy=
array([[   3,   52,    4,    3,  345,   52,    4, 4907,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0]])>
```

In [ ]:
```python
# Define the network
def build_model():
    input_ = Input(shape=(1,), dtype=tf.string)
    x = vectorize_layer(input_)
    x = Embedding(vocab_size, embedding_size, name="Embedding")(x)
    x = Bidirectional(
        LSTM(
            recurrent_size,
            return_sequences=False,
            dropout=dropout_rate,
            recurrent_dropout=dropout_rate,
        )
    )(x)
    x = Dense(hidden_size, activation="relu")(x)
    x = Dropout(dropout_rate)(x)
    output_ = Dense(nb_classes, activation="softmax", dtype=tf.float64)(x)
    model = Model(input_, output_)
    return model


model = build_model()
```

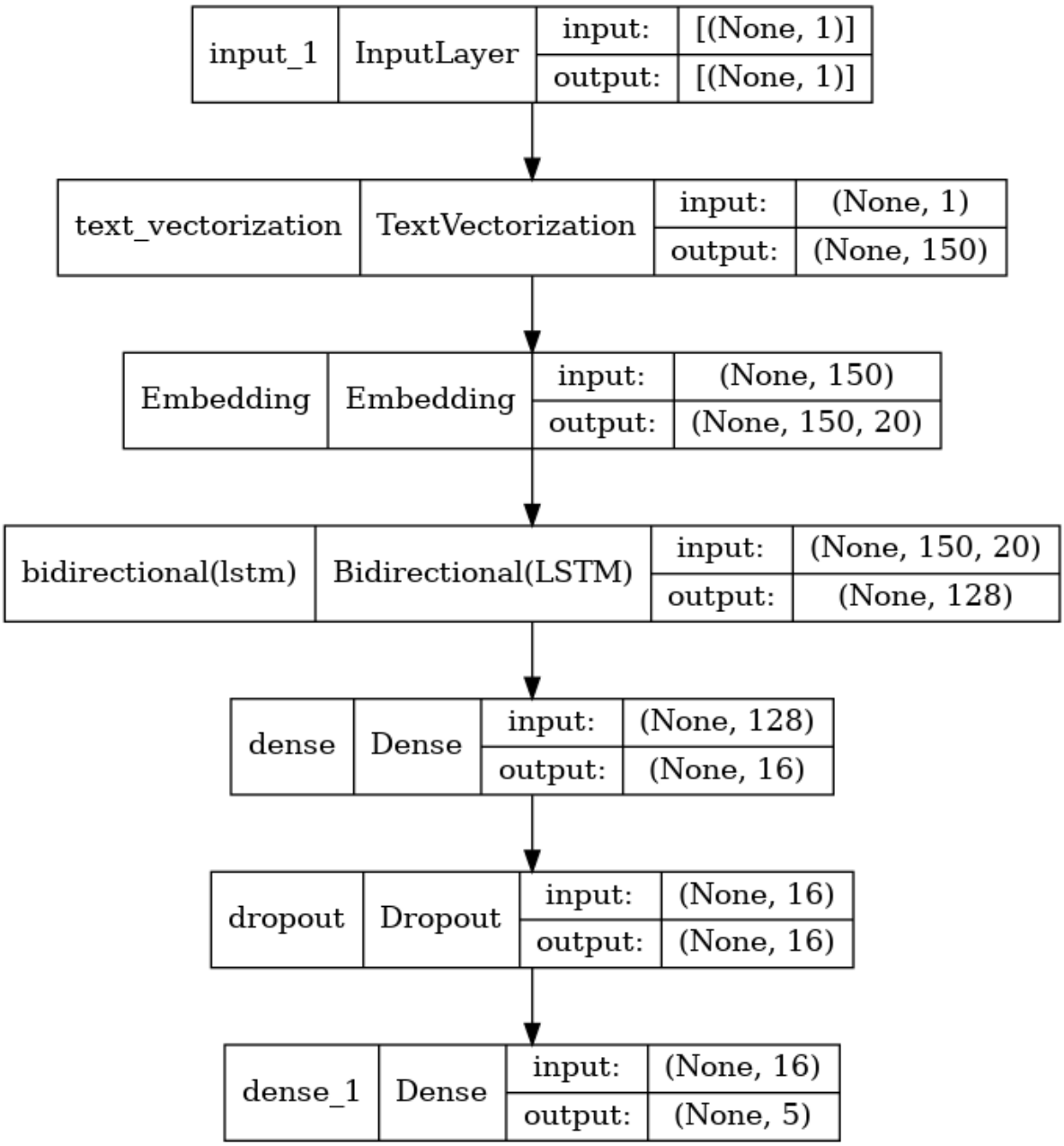
In [ ]:
```python
# summarize the model
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1)]               0

 text_vectorization (TextVec  (None, 150)              0
 torization)

 Embedding (Embedding)       (None, 150, 20)           200000

 bidirectional (Bidirectiona  (None, 128)              43520
 l)

 dense (Dense)               (None, 16)                2064

 dropout (Dropout)           (None, 16)                0

 dense_1 (Dense)             (None, 5)                 85

=================================================================
Total params: 245,669
Trainable params: 245,669
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
plot_model(model, show_shapes=True, show_layer_names=True, to_file="05-mode
```

Out[ ]:



In [ ]:

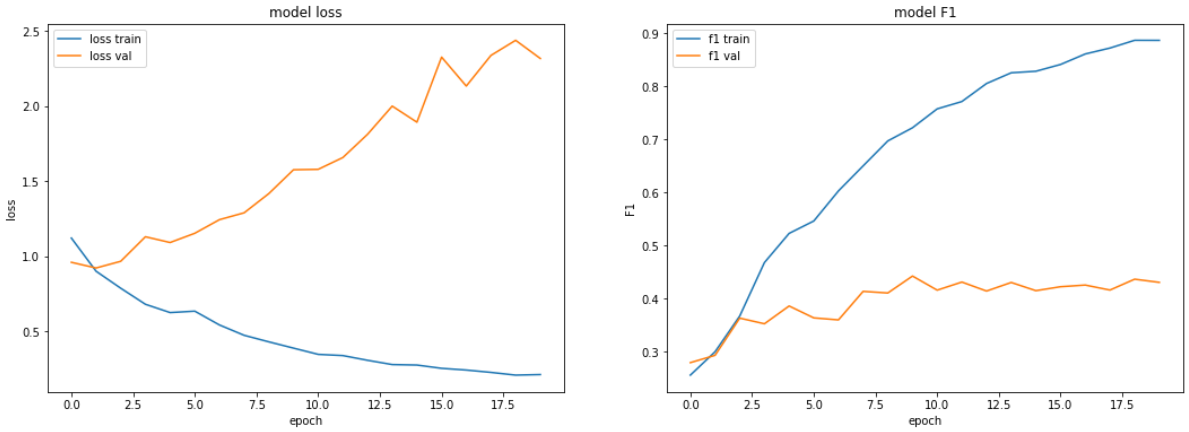
```
# Compile the model
f1 = F1Score(num_classes=nb_classes, average="macro", threshold=None)
op = Adam(learning_rate=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=1e
model.compile(optimizer=op, loss="categorical_crossentropy", metrics=[f1])
```

In [ ]:

```
# fit model using ealy stopping
es = EarlyStopping(
    monitor="val_f1_score",
    mode="max",
    patience=10,
    restore_best_weights=True,
    verbose=1,
)
history = model.fit(
    x=X_train,
    y=y_train,
    validation_data=(X_val, y_val),
    epochs=4000,
    callbacks=[es],
    verbose=1,
)
```

```
Epoch 1/4000
157/157 [==============================] - 44s 245ms/step - loss: 1.1212 -
f1_score: 0.2560 - val_loss: 0.9598 - val_f1_score: 0.2795
Epoch 2/4000
157/157 [==============================] - 36s 228ms/step - loss: 0.9005 -
f1_score: 0.2999 - val_loss: 0.9218 - val_f1_score: 0.2932
Epoch 3/4000
157/157 [==============================] - 36s 229ms/step - loss: 0.7872 -
f1_score: 0.3672 - val_loss: 0.9666 - val_f1_score: 0.3633
Epoch 4/4000
157/157 [==============================] - 35s 222ms/step - loss: 0.6803 -
f1_score: 0.4678 - val_loss: 1.1299 - val_f1_score: 0.3526
Epoch 5/4000
157/157 [==============================] - 36s 233ms/step - loss: 0.6251 -
f1_score: 0.5227 - val_loss: 1.0915 - val_f1_score: 0.3861
Epoch 6/4000
157/157 [==============================] - 37s 234ms/step - loss: 0.6345 -
f1_score: 0.5461 - val_loss: 1.1530 - val_f1_score: 0.3637
Epoch 7/4000
157/157 [==============================] - 38s 244ms/step - loss: 0.5426 -
f1_score: 0.6027 - val_loss: 1.2437 - val_f1_score: 0.3599
Epoch 8/4000
157/157 [==============================] - 38s 239ms/step - loss: 0.4738 -
f1_score: 0.6499 - val_loss: 1.2894 - val_f1_score: 0.4136
Epoch 9/4000
157/157 [==============================] - 39s 246ms/step - loss: 0.4303 -
f1_score: 0.6969 - val_loss: 1.4173 - val_f1_score: 0.4106
Epoch 10/4000
157/157 [==============================] - 34s 219ms/step - loss: 0.3884 -
f1_score: 0.7216 - val_loss: 1.5760 - val_f1_score: 0.4425
Epoch 11/4000
157/157 [==============================] - 37s 235ms/step - loss: 0.3469 -
f1_score: 0.7572 - val_loss: 1.5787 - val_f1_score: 0.4159
Epoch 12/4000
157/157 [==============================] - 34s 220ms/step - loss: 0.3388 -
f1_score: 0.7710 - val_loss: 1.6574 - val_f1_score: 0.4312
Epoch 13/4000
157/157 [==============================] - 33s 213ms/step - loss: 0.3074 -
f1_score: 0.8049 - val_loss: 1.8120 - val_f1_score: 0.4142
Epoch 14/4000
157/157 [==============================] - 34s 214ms/step - loss: 0.2789 -
f1_score: 0.8251 - val_loss: 1.9997 - val_f1_score: 0.4304
Epoch 15/4000
157/157 [==============================] - 33s 213ms/step - loss: 0.2758 -
f1_score: 0.8280 - val_loss: 1.8926 - val_f1_score: 0.4149
Epoch 16/4000
157/157 [==============================] - 33s 208ms/step - loss: 0.2539 -
f1_score: 0.8408 - val_loss: 2.3260 - val_f1_score: 0.4224
Epoch 17/4000
157/157 [==============================] - 33s 209ms/step - loss: 0.2424 -
f1_score: 0.8605 - val_loss: 2.1332 - val_f1_score: 0.4255
Epoch 18/4000
157/157 [==============================] - 33s 209ms/step - loss: 0.2268 -
f1_score: 0.8717 - val_loss: 2.3372 - val_f1_score: 0.4161
Epoch 19/4000
157/157 [==============================] - 33s 211ms/step - loss: 0.2087 -
f1_score: 0.8863 - val_loss: 2.4378 - val_f1_score: 0.4366
Epoch 20/4000
157/157 [==============================] - ETA: 0s - loss: 0.2125 - f1_scor
e: 0.8862Restoring model weights from the end of the best epoch: 10.
157/157 [==============================] - 33s 212ms/step - loss: 0.2125 -
f1_score: 0.8862 - val_loss: 2.3167 - val_f1_score: 0.4306
Epoch 00020: early stopping
```

In [ ]:
```
# plot history
def babysit(history):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

    # summarize history for loss
    ax1.plot(history.history["loss"])
    ax1.plot(history.history["val_loss"])
    ax1.set_title("model loss")
    ax1.set_ylabel("loss")
    ax1.set_xlabel("epoch")
    ax1.legend(["loss train", "loss val"], loc="best")

    # summarize history for loss
    ax2.plot(history.history["f1_score"])
    ax2.plot(history.history["val_f1_score"])
    ax2.set_title("model F1")
    ax2.set_ylabel("F1")
    ax2.set_xlabel("epoch")
    ax2.legend(["f1 train", "f1 val"], loc="best")

    plt.show()


babysit(history)
```



In [ ]:
```
# Evaluate the model
f1.update_state(y_test, model.predict(X_test))
print(f"F1: {f1.result().numpy()}")
```

```
F1: 0.42998647689819336
```

To do student

1. Understand the code
2. Play with LSTM model for sentiment analysis
   - Replace LSTM by BI-LSTM
   - Use stacked LSTM or BI-LSTM * Use all hidden state and average it

If you want to go further

If you are interested in the subject, current networks for sentiment prediction combine a part with recurrent networks (LTSM) to capture long dependencies and a part with convolution (CNN) to capture short dependencies. This resarch paper or this one describe some accurate networks for sentiment analysis.

Here, another paper that gives you some indications to go further: Attention, CNN and what not for Text Classification

You will see next week the CNN with Diane. So there is no need to use them today.

Attention with LSTM network

In [ ]:

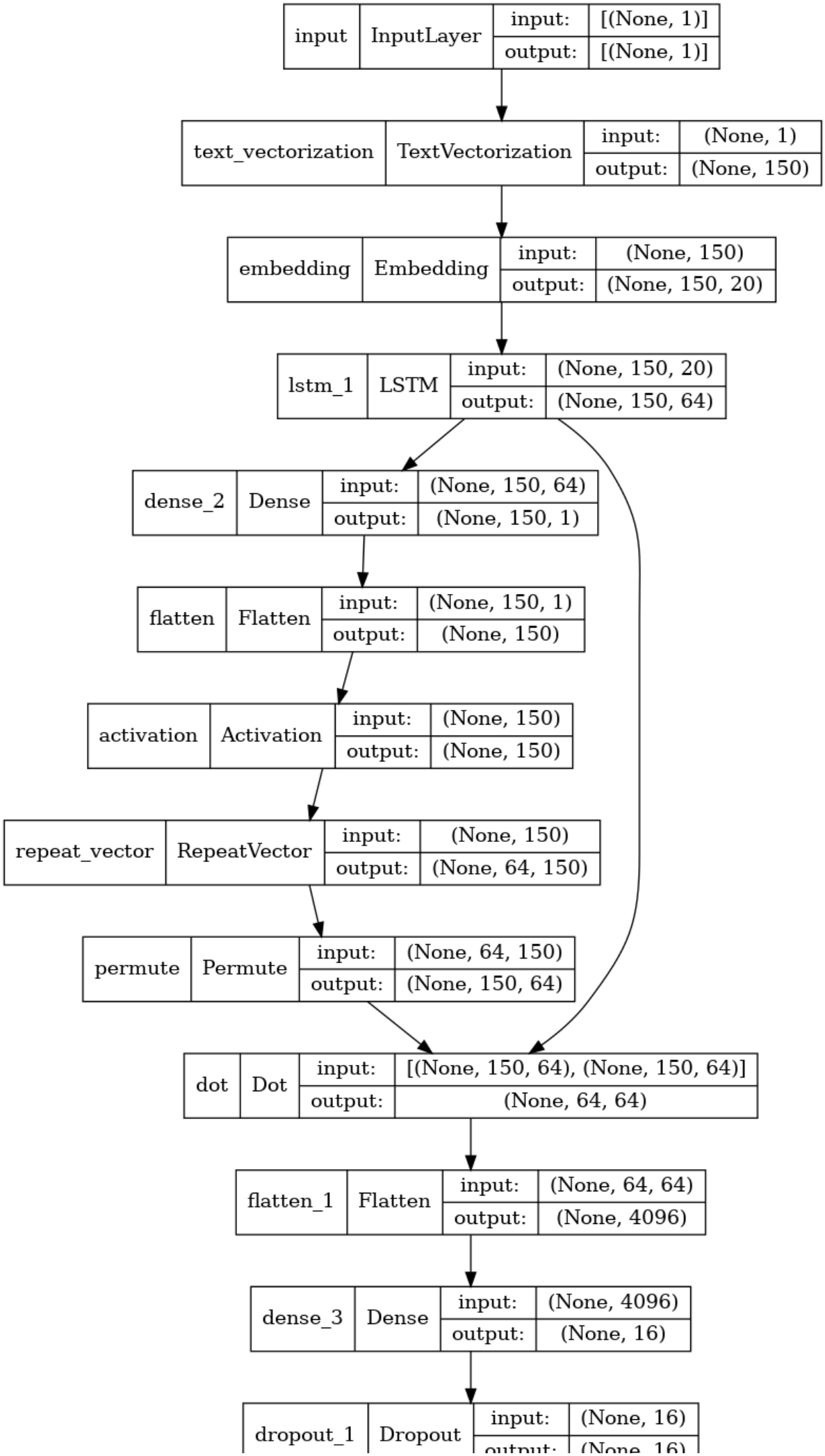
```
# ------------------------------------------------------
# MODEL BUILDING
# ------------------------------------------------------
def build_model():
    # Input: a review
    input_ = Input(shape=(1,), name="input", dtype=tf.string)

    # Transform the review in a list of tokenID
    vect = vectorize_layer(input_)

    # Keras embedding
    embedding = Embedding(
        input_dim=vocab_size,
        output_dim=embedding_size,
        weights=None,  # Without pre-learning
        trainable=True,  # Trainable
        name="embedding",
    )(vect)

    # You can try also a Bidirectionnel cell
    rnn = LSTM(
        recurrent_size,
        return_sequences=True,
        return_state=False,
        dropout=dropout_rate,
        recurrent_dropout=dropout_rate,
    )(embedding)

    # In the case of LSTM, there are two internal states
    #      the hidden state, usually denoted by h,
    #      the cell state usually denoted by c
    # The tuple (c, h) is the internal state of a LSTM
    # return_sequences=True gives you the hidden state (h) of the LSTM for
    # used in combination with return_state=True, you will only get the tup

    # Attention layer
    attention = Dense(1, activation="tanh")(rnn)
    attention = Flatten()(attention)
    attention = Activation("softmax")(attention)

    # Pour pouvoir faire la multiplication (scalair/vecteur KERAS)
    attention = RepeatVector(recurrent_size)(attention)  # NORMAL RNN
    attention = Permute([2, 1])(attention)

    # Application de l'attention sur la sortie du RNN
    sent_representation = Dot(axes=1, normalize=False)([rnn, attention])

    # Flatten pour entrer dans le Dense
    flatten = Flatten()(sent_representation)

    # Dense pour la classification avec 1 couche cachee
    hidden_dense = Dense(hidden_size, activation="relu")(flatten)
    hidden_dense = Dropout(dropout_rate)(hidden_dense)
```

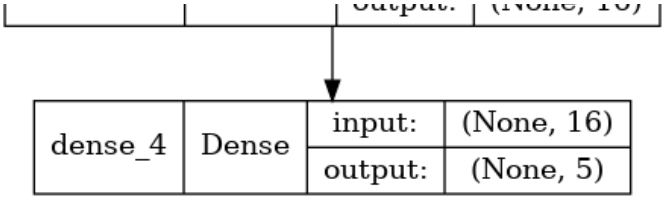
```
    # Classification et ouput
    output_ = Dense(nb_classes, activation="softmax")(hidden_dense)

    # Build  model
    model = Model(inputs=input_, outputs=output_)

    return model


model = build_model()
```

In [ ]:

```
# Plot model
plot_model(
    model=model, show_shapes=True, show_layer_names=True, to_file="LSTM_wit
)
```

Out[ ]:


input | InputLayer | input: [(None, 1)] | output: [(None, 1)]
text_vectorization | TextVectorization | input: (None, 1) | output: (None, 150)
embedding | Embedding | input: (None, 150) | output: (None, 150, 20)
lstm_1 | LSTM | input: (None, 150, 20) | output: (None, 150, 64)
dense_2 | Dense | input: (None, 150, 64) | output: (None, 150, 1)
flatten | Flatten | input: (None, 150, 1) | output: (None, 150)
activation | Activation | input: (None, 150) | output: (None, 150)
repeat_vector | RepeatVector | input: (None, 150) | output: (None, 64, 150)
permute | Permute | input: (None, 64, 150) | output: (None, 150, 64)
dot | Dot | input: [(None, 150, 64), (None, 150, 64)] | output: (None, 64, 64)
flatten_1 | Flatten | input: (None, 64, 64) | output: (None, 4096)
dense_3 | Dense | input: (None, 4096) | output: (None, 16)
dropout_1 | Dropout | input: (None, 16)



In [ ]:

```
# ----------------------------------------------------
# MODEL BUILDING
# ----------------------------------------------------
def build_hypermodel(hp):
    # Input: a review
    input_ = Input(shape=(1,), name="input", dtype=tf.string)

    # Transform the review in a list of tokenID
    vect = vectorize_layer(input_)

    # Keras embedding
    embedding = Embedding(
        input_dim=vocab_size,
        output_dim=embedding_size,
        weights=None,  # Without pre-learning
        trainable=True,  # Trainable
        name="embedding",
    )(vect)

    # You can try also a Bidirectionnel cell
    rnn = LSTM(
        recurrent_size,
        return_sequences=True,
        return_state=False,
        dropout=dropout_rate,
        recurrent_dropout=dropout_rate,
    )(embedding)

    # In the case of LSTM, there are two internal states
    #      the hidden state, usually denoted by h,
    #      the cell state usually denoted by c
    # The tuple (c, h) is the internal state of a LSTM
    # return_sequences=True gives you the hidden state (h) of the LSTM for
    # used in combination with return_state=True, you will only get the tup

    # Attention layer
    attention = Dense(1, activation="tanh")(rnn)
    attention = Flatten()(attention)
    attention = Activation("softmax")(attention)

    # Pour pouvoir faire la multiplication (scalair/vecteur KERAS)
    attention = RepeatVector(recurrent_size)(attention)  # NORMAL RNN
    attention = Permute([2, 1])(attention)

    # Application de l'attention sur la sortie du RNN
    sent_representation = Dot(axes=1, normalize=False)([rnn, attention])

    # Flatten pour entrer dans le Dense
    flatten = Flatten()(sent_representation)

    # Dense pour la classification avec 1 couche cachee
    hp_hidden_size = hp.Choice('units', values=[2**power for power in range
    hp_dropout_rate = hp.Choice('rate', values=[0.1 + k*0.05 for k in range

    hidden_dense = Dense(hp_hidden_size, activation="relu")(flatten)
    hidden_dense = Dropout(hp_dropout_rate)(hidden_dense)
```

```
    # Classification et ouput
    output_ = Dense(nb_classes, activation="softmax")(hidden_dense)

    # Build  model
    model = Model(inputs=input_, outputs=output_)

    # Compile the model
    f1 = F1Score(num_classes=nb_classes, average="macro", threshold=0.5)

    hp_learning_rate = hp.Choice('learning_rate', values=[0.005, 0.001, 0.0
    hp_beta_1 = hp.Choice('beta_1', values=[0.8, 0.9, 0.99, 0.999])
    hp_beta_2 = hp.Choice('beta_2', values=[0.99, 0.999, 0.9999])

    op = Adam(learning_rate=hp_learning_rate, beta_1=hp_beta_1, beta_2=hp_b
    model.compile(optimizer=op, loss="categorical_crossentropy", metrics=[f

    return model
```

In [ ]:

```
# Compile the model
f1 = F1Score(num_classes=nb_classes, average="macro", threshold=0.5)
op = Adam(learning_rate=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=1e
model.compile(optimizer=op, loss="categorical_crossentropy", metrics=[f1])
```

In [ ]:

```
tuner = kt.Hyperband(
    hypermodel=build_hypermodel,
    objective=kt.Objective("val_f1_score", direction="max"),
    max_epochs=20,
    factor=3,
)

# fit model using ealy stopping
es = EarlyStopping(
    monitor="val_f1_score",
    mode="max",
    patience=20,
    restore_best_weights=True,
    verbose=2,
)

tuner.search(X_train, y_train, epochs=20, validation_data=(X_val, y_val), c
```

```
Trial 30 Complete [00h 08m 00s]
val_f1_score: 0.4582522511482239

Best val_f1_score So Far: 0.4582522511482239
Total elapsed time: 01h 39m 31s
INFO:tensorflow:Oracle triggered exit
```

In [ ]:

```
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
best_hps.values
```

Out[ ]:
```
{'units': 32,
 'rate': 0.25,
 'learning_rate': 0.005,
 'beta_1': 0.999,
 'beta_2': 0.999,
 'tuner/epochs': 20,
 'tuner/initial_epoch': 0,
 'tuner/bracket': 0,
 'tuner/round': 0}
```

In [ ]:
```
hp_model = tuner.hypermodel.build(best_hps)
```

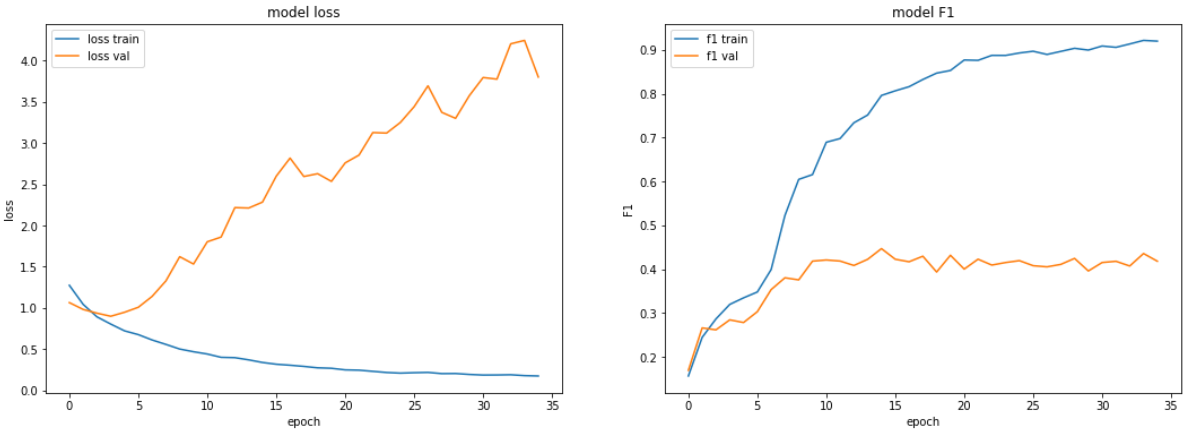
In [ ]:
```
history = hp_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=4000,
    callbacks=[es],
    verbose=1,
)
```

```
Epoch 1/4000
157/157 [==============================] - 26s 150ms/step - loss: 1.2740 -
f1_score: 0.1569 - val_loss: 1.0647 - val_f1_score: 0.1698
Epoch 2/4000
157/157 [==============================] - 23s 147ms/step - loss: 1.0433 -
f1_score: 0.2448 - val_loss: 0.9819 - val_f1_score: 0.2664
Epoch 3/4000
157/157 [==============================] - 23s 149ms/step - loss: 0.8923 -
f1_score: 0.2869 - val_loss: 0.9368 - val_f1_score: 0.2620
Epoch 4/4000
157/157 [==============================] - 23s 147ms/step - loss: 0.8045 -
f1_score: 0.3199 - val_loss: 0.8993 - val_f1_score: 0.2848
Epoch 5/4000
157/157 [==============================] - 24s 150ms/step - loss: 0.7224 -
f1_score: 0.3349 - val_loss: 0.9478 - val_f1_score: 0.2787
Epoch 6/4000
157/157 [==============================] - 23s 149ms/step - loss: 0.6768 -
f1_score: 0.3483 - val_loss: 1.0084 - val_f1_score: 0.3034
Epoch 7/4000
157/157 [==============================] - 24s 153ms/step - loss: 0.6117 -
f1_score: 0.3994 - val_loss: 1.1404 - val_f1_score: 0.3537
Epoch 8/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.5582 -
f1_score: 0.5226 - val_loss: 1.3284 - val_f1_score: 0.3806
Epoch 9/4000
157/157 [==============================] - 23s 149ms/step - loss: 0.5019 -
f1_score: 0.6048 - val_loss: 1.6217 - val_f1_score: 0.3759
Epoch 10/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.4692 -
f1_score: 0.6158 - val_loss: 1.5323 - val_f1_score: 0.4186
Epoch 11/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.4416 -
f1_score: 0.6895 - val_loss: 1.8025 - val_f1_score: 0.4212
Epoch 12/4000
157/157 [==============================] - 24s 153ms/step - loss: 0.4013 -
f1_score: 0.6979 - val_loss: 1.8593 - val_f1_score: 0.4188
Epoch 13/4000
157/157 [==============================] - 23s 148ms/step - loss: 0.3973 -
f1_score: 0.7338 - val_loss: 2.2167 - val_f1_score: 0.4087
Epoch 14/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.3708 -
f1_score: 0.7515 - val_loss: 2.2115 - val_f1_score: 0.4227
Epoch 15/4000
157/157 [==============================] - 23s 149ms/step - loss: 0.3392 -
f1_score: 0.7961 - val_loss: 2.2829 - val_f1_score: 0.4471
Epoch 16/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.3178 -
f1_score: 0.8066 - val_loss: 2.5977 - val_f1_score: 0.4229
Epoch 17/4000
157/157 [==============================] - 24s 153ms/step - loss: 0.3057 -
f1_score: 0.8161 - val_loss: 2.8169 - val_f1_score: 0.4171
Epoch 18/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.2915 -
f1_score: 0.8324 - val_loss: 2.5934 - val_f1_score: 0.4298
Epoch 19/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.2742 -
f1_score: 0.8470 - val_loss: 2.6276 - val_f1_score: 0.3937
Epoch 20/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.2687 -
f1_score: 0.8530 - val_loss: 2.5345 - val_f1_score: 0.4320
Epoch 21/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.2495 -
f1_score: 0.8769 - val_loss: 2.7590 - val_f1_score: 0.4004
Epoch 22/4000
```

```
157/157 [==============================] - 24s 152ms/step - loss: 0.2457 -
f1_score: 0.8761 - val_loss: 2.8531 - val_f1_score: 0.4230
Epoch 23/4000
157/157 [==============================] - 24s 153ms/step - loss: 0.2317 -
f1_score: 0.8874 - val_loss: 3.1263 - val_f1_score: 0.4096
Epoch 24/4000
157/157 [==============================] - 24s 153ms/step - loss: 0.2176 -
f1_score: 0.8873 - val_loss: 3.1201 - val_f1_score: 0.4154
Epoch 25/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.2099 -
f1_score: 0.8929 - val_loss: 3.2489 - val_f1_score: 0.4197
Epoch 26/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.2150 -
f1_score: 0.8968 - val_loss: 3.4406 - val_f1_score: 0.4082
Epoch 27/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.2187 -
f1_score: 0.8895 - val_loss: 3.6931 - val_f1_score: 0.4056
Epoch 28/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.2038 -
f1_score: 0.8965 - val_loss: 3.3717 - val_f1_score: 0.4111
Epoch 29/4000
157/157 [==============================] - 24s 151ms/step - loss: 0.2051 -
f1_score: 0.9035 - val_loss: 3.2988 - val_f1_score: 0.4248
Epoch 30/4000
157/157 [==============================] - 24s 150ms/step - loss: 0.1940 -
f1_score: 0.8995 - val_loss: 3.5741 - val_f1_score: 0.3961
Epoch 31/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.1868 -
f1_score: 0.9087 - val_loss: 3.7937 - val_f1_score: 0.4154
Epoch 32/4000
157/157 [==============================] - 24s 156ms/step - loss: 0.1876 -
f1_score: 0.9057 - val_loss: 3.7737 - val_f1_score: 0.4182
Epoch 33/4000
157/157 [==============================] - 24s 152ms/step - loss: 0.1903 -
f1_score: 0.9134 - val_loss: 4.2022 - val_f1_score: 0.4075
Epoch 34/4000
157/157 [==============================] - 24s 153ms/step - loss: 0.1796 -
f1_score: 0.9215 - val_loss: 4.2436 - val_f1_score: 0.4359
Epoch 35/4000
157/157 [==============================] - ETA: 0s - loss: 0.1753 - f1_scor
e: 0.9198Restoring model weights from the end of the best epoch: 15.
157/157 [==============================] - 24s 151ms/step - loss: 0.1753 -
f1_score: 0.9198 - val_loss: 3.7994 - val_f1_score: 0.4184
Epoch 00035: early stopping
```

In [ ]:

```
# plot history
babysit(history)
```



In [ ]:

```
# Evaluate the model
f1.update_state(y_test, hp_model.predict(X_test))
print(f"F1: {f1.result()}")
```

```
F1: 0.4235216975212097
```

Use Attentionnal model

In the cell below, we reproduce our existing model until the Activation layer only.

In [ ]:

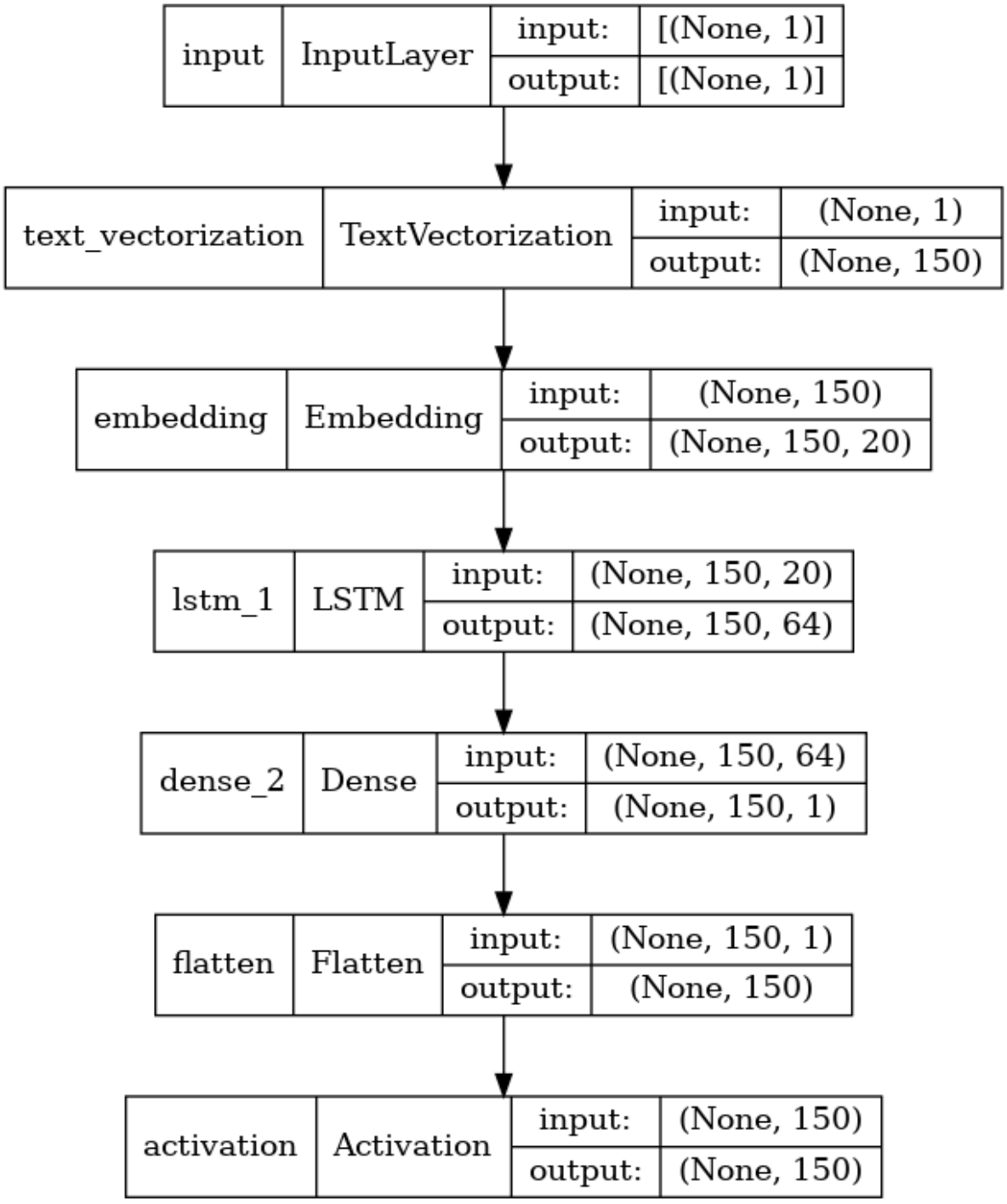
```
# ----------------------------------------------------------
# GET ATTENTION MODEL
# ----------------------------------------------------------
def get_attention_model(model):
    attention_layer_indice = 0
    for layer in model.layers:
        print(type(layer))
        if type(layer) is Activation:
            break
        else:
            attention_layer_indice += 1

    # Create an attention model
    return Model(
        inputs=model.layers[0].input,
        outputs=model.layers[attention_layer_indice].output,
    )
```

In [ ]:

```
# PLOT ATTENTION MODEL from classifier model with ATTENTION
attention_model = get_attention_model(model)
plot_model(
    attention_model,
    show_shapes=True,
    show_layer_names=True,
    to_file="model_get_attention.png",
)
```

```
<class 'keras.engine.input_layer.InputLayer'>
<class 'keras.layers.preprocessing.text_vectorization.TextVectorization'>
<class 'keras.layers.embeddings.Embedding'>
<class 'keras.layers.recurrent_v2.LSTM'>
<class 'keras.layers.core.dense.Dense'>
<class 'keras.layers.core.flatten.Flatten'>
<class 'keras.layers.core.activation.Activation'>
```

Out[ ]:



In [ ]:

```
# ------------------------------------------------------
# GET ATTENTION
# ------------------------------------------------------
attentions = attention_model.predict(X_val[0])
attentions
```

Out[ ]:
```
array([[0.00669012, 0.00667176, 0.00666747, 0.00666548, 0.00666473,
        0.00666467, 0.00666497, 0.00666542, 0.00666588, 0.00666631,
        0.00666666, 0.00666693, 0.00666713, 0.00666726, 0.00666734,
        0.00666737, 0.00666736, 0.00666733, 0.00666728, 0.00666722,
        0.00666715, 0.00666708, 0.006667  , 0.00666693, 0.00666687,
        0.0066668 , 0.00666675, 0.0066667 , 0.00666665, 0.00666661,
        0.00666658, 0.00666655, 0.00666653, 0.00666651, 0.00666649,
        0.00666648, 0.00666647, 0.00666646, 0.00666645, 0.00666644,
        0.00666644, 0.00666644, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643,
        0.00666643, 0.00666643, 0.00666643, 0.00666643, 0.00666643]],
      dtype=float32)
```

Now we measure the attention given by our model to some sample words. The model gives quite some attention to "pretty" and "good". This makes sense since it is a meaningful indicator which hints that the review will be positive. \ Had the terms been more vague, the mdoel would have (or at least should have) given them less attention.

In [ ]:
```
# -------------------------------------------------------
# GET ATTENTION for each WORD
# -------------------------------------------------------
from sklearn import preprocessing


def get_attention(X, y, prediction, attention, N=5):
    # normalize attention (without the padding part)
    normalized_attention = preprocessing.QuantileTransformer().fit_transfor

    results = []
    for i, (X_, y_, p_, a_) in enumerate(zip(X, y, prediction, normalized_a
        if i > N:
            break
        # build result
        result_entry = {}
        result_entry["prediction"] = (np.argmax(y_), np.argmax(p_))
        result_entry["original"] = np.asscalar(X_)
        result_entry["sentence"] = []
        for j, word in enumerate(vectorize_layer(X_).numpy().flatten().toli
            word_obj = {}
            if word == 0:
                break
            word_obj[vectorize_layer.get_vocabulary()[word]] = a_[j].item()
            result_entry["sentence"].append(word_obj)
```

```
        results += [result_entry]
    return results


sentences_with_attention = get_attention(
    X_val, y_val, model.predict(X_val), attention_model.predict(X_val), 10
)
sentences_with_attention[0]
```

```
/tmp/ipykernel_29598/4255116920.py:18: DeprecationWarning:

np.asscalar(a) is deprecated since NumPy v1.16, use a.item() instead
```

Out[ ]:

```
{'prediction': (0, 4),
 'original': 'Pretty good',
 'sentence': [{'pretty': 0.10360360145568848}, {'good': 0.06156156212091446}]}
```

For a clear explanation, we display in green the words with attention > 0.75 (meaning the model gives them a lot of attention), in red the words with attention < 0.25 (meaning the model gives them little attention), and in grey the other words. \ The `[UNK]` words are words that are unknown to the model (*i.e.* they are not part of the vocabulary). If we increased the vocabulary size (and therefore the model complexity), some of them would stop being unknowned by our model.

In [ ]:

```
# convert prediction with attention to colored text
from termcolor import colored


def print_text(sentences_with_attention):
    threshold = 0.75
    classes = []
    print(colored("In green, the most important word\n\n", "green", attrs=[

    for i, sentence in enumerate(sentences_with_attention):
        # Retrieve the class of this sentence
        # print(sentence)
        original_class, predicted_class = sentence["prediction"]
        # print(original_class, predicted_class)

        # Retrieve all the words and weights of this sentence
        words, weights = [], []
        # print("--", sentence['sentence'])
        for item in sentence["sentence"]:
            for word, weight in item.items():
                words.append(word)
                weights.append(float(weight))

        size = 0
        print(sentence["original"])
        for j, word in enumerate(words):
            if size != 0 and j != 0 and word != "," and word != ".":
                print(" ", end="")
            if weights[j] > threshold:
                print(colored(word, "green", attrs=["bold"]), end=" ")
            elif weights[j] < (1 - threshold):
                print(colored(word, "red", attrs=["bold"]), end=" ")
            else:
                print(colored(word, "grey"), end=" ")
            size += len(word) + 1
```

```
            if size > 80:
                print()
            size = 0

        print("\n")


print_text(sentences_with_attention)
```

In green, the most important word

Pretty good
pretty good

Excellent,very good Very fast
[UNK] good very **fast**

This was a replacement. We had a POS Samsung from Cincinnati Bell. That pho
ne was the worst thing we ever owned! This HTC seems to be built to last. I
t is small. Dont plan on rocking a lot of video. However, it is an Andriod
 and I think my wife likes it. It feels sturdy too. We bought it unlocked.
 All we had to do was insert the SIM card. It took about 2min to get it all
set up. Very nice! I would give it 4-stars but the internal memory will eve
ntually be an issue.
this was a replacement we had a pos samsung from [UNK] [UNK] that phone was
the **worst thing** we ever owned this htc seems to **be** built **to last it is smal
l dont plan on [UNK] a lot of video however it is an andriod and i think my
wife likes it it feels sturdy too we bought it unlocked all we had to do wa
s insert the sim card it took about [UNK] to get it all set up very nice i**
would **give it [UNK] but the internal memory will eventually be an issue**

Was used ! Had to return it
was used had to return it

Just loved the item, and it was just as the pictured stated, therefore i'm
 happy with my purchase, but i feel it being a Nokia it should of been able
to do a bit more.
just **loved the** item and **it** was just as the **pictured** stated **therefore** im **hap
py with my purchase but i feel it being a nokia it should of been** able **to d
o a bit more**

total confianza y satisfacción el un producto de excelente calidad el tiemp
o de entrega fue muy breve y estoy muy contento
total [UNK] **y** [UNK] el **un producto de** excelente calidad el tiempo de **entreg
a fue muy [UNK] y estoy muy contento**

Liberado ? Unlocked??
liberado unlocked

At first there were two problems with the product iPhone 4s unlocked I purc
hased . The WiFi tab was greyed out dim and I can't connect to Internet eve
n I have a sim card with high speed internet and Wi Fi at home. There is no
cellular data network tab on the phone to edit the APN. I called Wireless 2
go They told me they had never had this issue before and they were ready to
refund me and sent me a label to return the product. After consideration I
 decided not to return the product. I reset and erased all the the content
 and network and started activating the iPhone again. It was a hard time! F
inally the Wi Fi tab works and the Internet is connected. However when I br
owsed Google the Wi Fi stopped working and the screen displayed "could not
 activate cellular dater network" . I had to redo everything again. Totally
3 times. Now it is connected to Wi Fi and Internet but I don't know how lon
g this status can maintain. I even dare not to browse Google website on thi
s iphone. I think the shadow of the former carrier is still there even the
 phone is unlocked. Is there a way to find out the cellular data network ta
b and APN to solve this problem?Anyway I have to say that this iPhone 4s ha
s a good look and almost no scratches.
at first there were two **problems with the** product **iphone** 4s unlocked **i** purc
hased the wifi **tab was greyed out dim and** i **cant connect to internet even** i
have a sim **card with high speed internet** and wi **fi at home there is no cell
ular data network tab on the phone to edit the apn i called wireless 2 go t
hey told me they had** never **had this** issue **before and they were ready** to **ref
und me and** sent **me a label** to return the **product after consideration i deci**

```
ded not to return the product i reset and erased all the the content and ne
twork and started activating the iphone again it was a hard time finally th
e wi fi tab works and the internet is connected however when i [UNK] google
the wi fi stopped working and the screen displayed could not activate cellu
lar [UNK] network i had

Its ok
its ok

nice little phone .had to send it back when it stopped receiving calls.
nice little phone had to send it back when it stopped receiving calls

Very much like my mothers original phone. Just newer! Ease to use was the n
eeded aspect.
very much like my [UNK] original phone just newer ease to use was the neede
d aspect
```

Your work

TO DO Students

1. Before modifying the code, take the time to understand it well. We use here the global attentions mechanism only from an encoder since the network for sentiment analysis has no decoder part, only a classifier 1
2. Improve the f1 score for the **Attentional LSTM** model using BI-LSTM approach, better hyper-parameters and a better preprocessing (the same as in the previous step).
    - Take inspiration from the course slides to build an original architecture that you will describe
    - Use your Attention part in order to explain the words taken into account by the network to predict the sentiment.
3. **Upload on moodle**
    - **a clean, documented notebook** containing **your best LSTM attentional model**. The evaluation metric is the f1 score (macro avg).
    - You can build all sorts of architectures but only using the cells seen in class (i.e. in particular: **CNNs are not yet seen so you should not use them**).
    - It is of course possible / desirable to use keras tuner to get the best hyper-parameters.
    - This notebook will be evaluated and the grade will take into account the editorial quality of your text.
    - Any notebook containing more than 1 model will not be evaluated (score = 0 -> **You have to choose the best one**).

