ONE HOT ENCODING

Imports libraries

In [1]:
```
vocab_size = 100000  #number of unique words
embedding_dim = 16
max_length = 80
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 200000
```

https://www.actuia.com/keras/debuter-avec-le-modele-sequentiel-de-keras/

https://inside-machinelearning.com/en/efficient-sentences-embedding-visualization-tsne/#Preparing_the_data_-_GoEmotions

https://vitalflux.com/keras-categorical-cross-entropy-loss-function/ sur les loss : categorical_cross, binary_cross

In [2]:
```
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import seaborn as sns
from keras.utils import plot_model
import os
print(tf.__version__)
#!pip install -q tensorflow-datasets
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Activation, Embedding, Flatten, GlobalMaxPool1D, Dro
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
```

```
2.9.2
```

Load and transform the data

We concatenate the 3 dataframes so we can deal with a single one. We drop all the useless columns in the dataframe and only keep the sentiment and the tweet associated to it.

We also reset the index because we concatenated the 3 goemotions so without that we would have 3 rows with the same index

In [3]:
```
goemotions_1 = pd.read_csv('goemotions_1.csv') #Load the dataset
goemotions_2 = pd.read_csv('goemotions_2.csv') #Load the dataset
goemotions_3 = pd.read_csv('goemotions_3.csv') #Load the dataset
```

In [4]:
```
# concat 3 df
frames = [goemotions_1, goemotions_2, goemotions_3]
df = pd.concat(frames)
# drop others columns
df.drop(['id','author','subreddit','link_id','parent_id','created_utc','rater_id', '
df.reset_index(drop=True, inplace=True)
df.shape
```

Out[4]: (211225, 29)

We assign the following variables :

- sentences : the column "text" of df that refers to all the tweets in the dataframe
- labels : sentiment get dummies of the dataframe, all the sentiment columns so from the index 1 (index 0 being the text) to the last column) There are 28 sentiments in the dataframe

In [5]:
```
sentences = df['text']
# df of all the 28 emotions
labels = df.iloc[:, 1:]
#labels.shape
labels
```

Out[5]:

	admiration	amusement	anger	annoyance	approval	caring	confusion	curiosity	desire
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
211220	0	0	0	0	0	0	0	0	0
211221	0	0	0	0	0	1	0	0	0
211222	1	0	0	0	0	0	0	0	0
211223	0	0	1	0	0	0	0	0	0
211224	0	0	0	0	0	0	0	0	0

211225 rows × 28 columns

Create train and test

We associate to training_sentences the tweets for the training size so from 0 to 200 000

The testing_sentences is all the tweet from 200 000 to the end of the dataframe

The training_labels takes all the labels associated to the tweets for the training size (0 to 200 0000) The testing_labels takes the ramaining labels

In [6]:
```
# training_sentences select the text, testing_labels the labels associated
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]

# training_sentences[0] is the first tweet in training_sentences
```

We tokenize training_sentences and testing_sentences

In [7]:
```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

# make all the sentences of equal size, add zeros at the end of the sentencses to fi

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length, padding=paddi

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length, padding=padding
```

In [8]:
```
# Need this block to get it to work with TensorFlow 2.x
training_padded = np.array(training_padded)
training_labels = np.array(training_labels)
testing_padded = np.array(testing_padded)
testing_labels = np.array(testing_labels)
```

-- ACURACY METRIC ---------------------------------------

MODEL 1 loss = binary cross entropy

In [ ]:
```
```

Hyperparameters

The whole dataset is 211225 so we split inot a training size of 200 000 and the rest is the test set.

We set the maximum vocabulary size to 10 000 and the maximum lenght of a tweet to 150

703 is the longest tweet size

In [18]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(28 , activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 80, 16)            1600000

 global_average_pooling1d_4   (None, 16)               0
 (GlobalAveragePooling1D)

 dense_8 (Dense)             (None, 128)               2176

 dense_9 (Dense)             (None, 28)                3612

=================================================================
Total params: 1,605,788
Trainable params: 1,605,788
Non-trainable params: 0
_________________________________________________________________
```

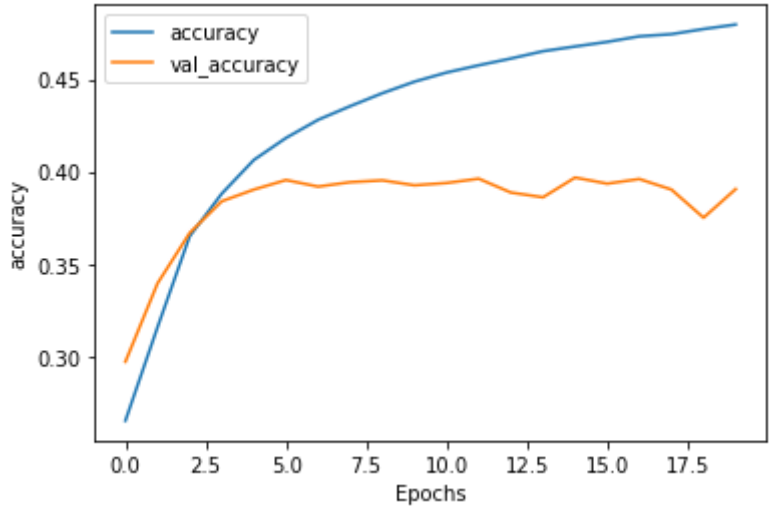
In [19]:
```
num_epochs = 20
history = model.fit(training_padded, training_labels, epochs=num_epochs, validation_
```

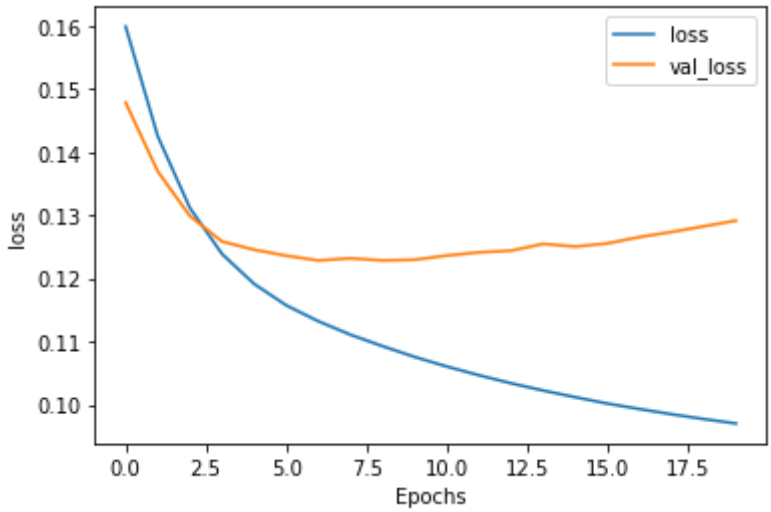
```
Epoch 1/20
6250/6250 - 17s - loss: 0.1598 - accuracy: 0.2652 - val_loss: 0.1478 - val_accuracy:
0.2971 - 17s/epoch - 3ms/step
Epoch 2/20
6250/6250 - 16s - loss: 0.1425 - accuracy: 0.3158 - val_loss: 0.1369 - val_accuracy:
0.3395 - 16s/epoch - 3ms/step
Epoch 3/20
6250/6250 - 16s - loss: 0.1311 - accuracy: 0.3650 - val_loss: 0.1299 - val_accuracy:
0.3668 - 16s/epoch - 3ms/step
Epoch 4/20
6250/6250 - 16s - loss: 0.1239 - accuracy: 0.3881 - val_loss: 0.1258 - val_accuracy:
0.3840 - 16s/epoch - 3ms/step
Epoch 5/20
6250/6250 - 17s - loss: 0.1191 - accuracy: 0.4063 - val_loss: 0.1245 - val_accuracy:
0.3903 - 17s/epoch - 3ms/step
Epoch 6/20
6250/6250 - 16s - loss: 0.1157 - accuracy: 0.4182 - val_loss: 0.1236 - val_accuracy:
0.3955 - 16s/epoch - 3ms/step
Epoch 7/20
6250/6250 - 16s - loss: 0.1132 - accuracy: 0.4281 - val_loss: 0.1228 - val_accuracy:
0.3919 - 16s/epoch - 3ms/step
Epoch 8/20
6250/6250 - 16s - loss: 0.1111 - accuracy: 0.4355 - val_loss: 0.1232 - val_accuracy:
```

```
0.3943 - 16s/epoch - 3ms/step
Epoch 9/20
6250/6250 - 17s - loss: 0.1093 - accuracy: 0.4425 - val_loss: 0.1228 - val_accuracy:
0.3953 - 17s/epoch - 3ms/step
Epoch 10/20
6250/6250 - 16s - loss: 0.1076 - accuracy: 0.4487 - val_loss: 0.1229 - val_accuracy:
0.3926 - 16s/epoch - 3ms/step
Epoch 11/20
6250/6250 - 16s - loss: 0.1060 - accuracy: 0.4537 - val_loss: 0.1236 - val_accuracy:
0.3939 - 16s/epoch - 3ms/step
Epoch 12/20
6250/6250 - 16s - loss: 0.1047 - accuracy: 0.4576 - val_loss: 0.1241 - val_accuracy:
0.3962 - 16s/epoch - 3ms/step
Epoch 13/20
6250/6250 - 16s - loss: 0.1034 - accuracy: 0.4612 - val_loss: 0.1244 - val_accuracy:
0.3887 - 16s/epoch - 3ms/step
Epoch 14/20
6250/6250 - 16s - loss: 0.1022 - accuracy: 0.4652 - val_loss: 0.1254 - val_accuracy:
0.3861 - 16s/epoch - 3ms/step
Epoch 15/20
6250/6250 - 16s - loss: 0.1012 - accuracy: 0.4678 - val_loss: 0.1250 - val_accuracy:
0.3968 - 16s/epoch - 3ms/step
Epoch 16/20
6250/6250 - 16s - loss: 0.1002 - accuracy: 0.4702 - val_loss: 0.1255 - val_accuracy:
0.3935 - 16s/epoch - 3ms/step
Epoch 17/20
6250/6250 - 17s - loss: 0.0993 - accuracy: 0.4732 - val_loss: 0.1265 - val_accuracy:
0.3960 - 17s/epoch - 3ms/step
Epoch 18/20
6250/6250 - 20s - loss: 0.0985 - accuracy: 0.4744 - val_loss: 0.1273 - val_accuracy:
0.3903 - 20s/epoch - 3ms/step
Epoch 19/20
6250/6250 - 18s - loss: 0.0977 - accuracy: 0.4772 - val_loss: 0.1282 - val_accuracy:
0.3751 - 18s/epoch - 3ms/step
Epoch 20/20
6250/6250 - 18s - loss: 0.0970 - accuracy: 0.4796 - val_loss: 0.1291 - val_accuracy:
0.3906 - 18s/epoch - 3ms/step
```

In [20]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```





Conclusion :

the model seems to behave well after each iteration of optimization but the loss stagnates and so the accuracy doesn't go beyond 0.4. The model tends to overfit a little bit at the end.

I try another way to predict the sentiment by having a label encoding instead of a one hot encoding process

MODEL 1 WITH DIFFERENT PARAMETERS vocab_size

---------- AUC METRIC ----------

MODEL 1

In [16]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(28 , activation='sigmoid')
```

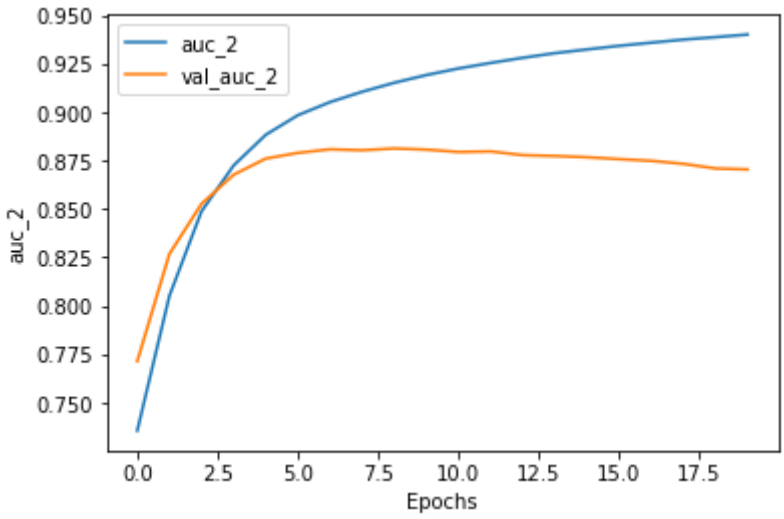
```
])
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=[tf.keras.metrics
model.summary()

num_epochs = 20
history = model.fit(training_padded, training_labels, epochs=num_epochs, validation_
```

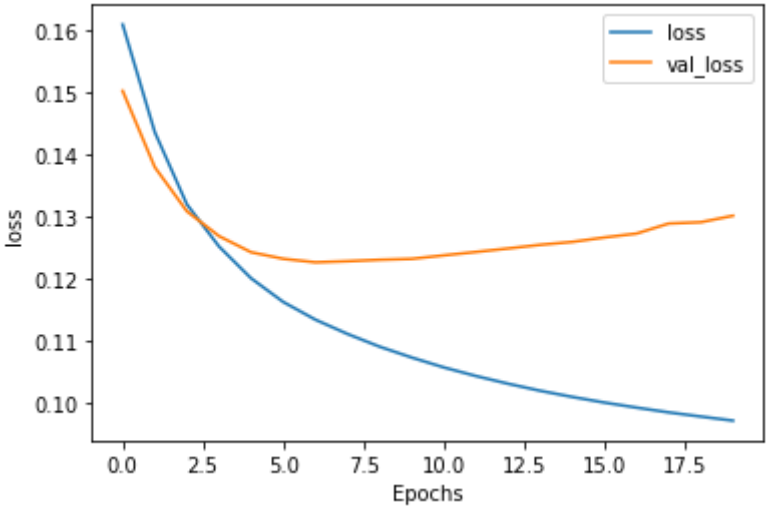
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 80, 16)            1600000

 global_average_pooling1d_3   (None, 16)               0
 (GlobalAveragePooling1D)

 dense_6 (Dense)             (None, 128)               2176

 dense_7 (Dense)             (None, 28)                3612

=================================================================
Total params: 1,605,788
Trainable params: 1,605,788
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
6250/6250 - 21s - loss: 0.1610 - auc_2: 0.7355 - val_loss: 0.1503 - val_auc_2: 0.771
4 - 21s/epoch - 3ms/step
Epoch 2/20
6250/6250 - 21s - loss: 0.1437 - auc_2: 0.8051 - val_loss: 0.1380 - val_auc_2: 0.826
7 - 21s/epoch - 3ms/step
Epoch 3/20
6250/6250 - 20s - loss: 0.1320 - auc_2: 0.8488 - val_loss: 0.1309 - val_auc_2: 0.852
6 - 20s/epoch - 3ms/step
Epoch 4/20
6250/6250 - 20s - loss: 0.1252 - auc_2: 0.8724 - val_loss: 0.1269 - val_auc_2: 0.867
6 - 20s/epoch - 3ms/step
Epoch 5/20
6250/6250 - 20s - loss: 0.1201 - auc_2: 0.8882 - val_loss: 0.1243 - val_auc_2: 0.876
0 - 20s/epoch - 3ms/step
Epoch 6/20
6250/6250 - 20s - loss: 0.1162 - auc_2: 0.8984 - val_loss: 0.1232 - val_auc_2: 0.878
9 - 20s/epoch - 3ms/step
Epoch 7/20
6250/6250 - 20s - loss: 0.1134 - auc_2: 0.9051 - val_loss: 0.1226 - val_auc_2: 0.880
8 - 20s/epoch - 3ms/step
Epoch 8/20
6250/6250 - 20s - loss: 0.1111 - auc_2: 0.9105 - val_loss: 0.1228 - val_auc_2: 0.880
3 - 20s/epoch - 3ms/step
Epoch 9/20
6250/6250 - 20s - loss: 0.1090 - auc_2: 0.9151 - val_loss: 0.1231 - val_auc_2: 0.881
2 - 20s/epoch - 3ms/step
Epoch 10/20
6250/6250 - 20s - loss: 0.1073 - auc_2: 0.9191 - val_loss: 0.1232 - val_auc_2: 0.880
6 - 20s/epoch - 3ms/step
Epoch 11/20
6250/6250 - 20s - loss: 0.1057 - auc_2: 0.9225 - val_loss: 0.1238 - val_auc_2: 0.879
4 - 20s/epoch - 3ms/step
Epoch 12/20
6250/6250 - 20s - loss: 0.1043 - auc_2: 0.9253 - val_loss: 0.1243 - val_auc_2: 0.879
7 - 20s/epoch - 3ms/step
Epoch 13/20
6250/6250 - 20s - loss: 0.1030 - auc_2: 0.9280 - val_loss: 0.1249 - val_auc_2: 0.877
8 - 20s/epoch - 3ms/step
```

```
Epoch 14/20
6250/6250 - 20s - loss: 0.1019 - auc_2: 0.9304 - val_loss: 0.1255 - val_auc_2: 0.877
3 - 20s/epoch - 3ms/step
Epoch 15/20
6250/6250 - 20s - loss: 0.1009 - auc_2: 0.9323 - val_loss: 0.1259 - val_auc_2: 0.876
7 - 20s/epoch - 3ms/step
Epoch 16/20
6250/6250 - 20s - loss: 0.1000 - auc_2: 0.9342 - val_loss: 0.1267 - val_auc_2: 0.875
7 - 20s/epoch - 3ms/step
Epoch 17/20
6250/6250 - 21s - loss: 0.0992 - auc_2: 0.9358 - val_loss: 0.1273 - val_auc_2: 0.874
8 - 21s/epoch - 3ms/step
Epoch 18/20
6250/6250 - 21s - loss: 0.0984 - auc_2: 0.9374 - val_loss: 0.1289 - val_auc_2: 0.873
3 - 21s/epoch - 3ms/step
Epoch 19/20
6250/6250 - 20s - loss: 0.0977 - auc_2: 0.9387 - val_loss: 0.1291 - val_auc_2: 0.870
9 - 20s/epoch - 3ms/step
Epoch 20/20
6250/6250 - 20s - loss: 0.0971 - auc_2: 0.9400 - val_loss: 0.1302 - val_auc_2: 0.870
4 - 20s/epoch - 3ms/step
```

In [17]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "auc_2")
plot_graphs(history, "loss")
```





The loss decreases and the accuracy increases so the training process is efficient. There is no overfitting

MODEL 1 with dropout

In [14]:

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(28 , activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=[tf.keras.metrics
model.summary()

num_epochs = 20
history = model.fit(training_padded, training_labels, epochs=num_epochs, validation_
```

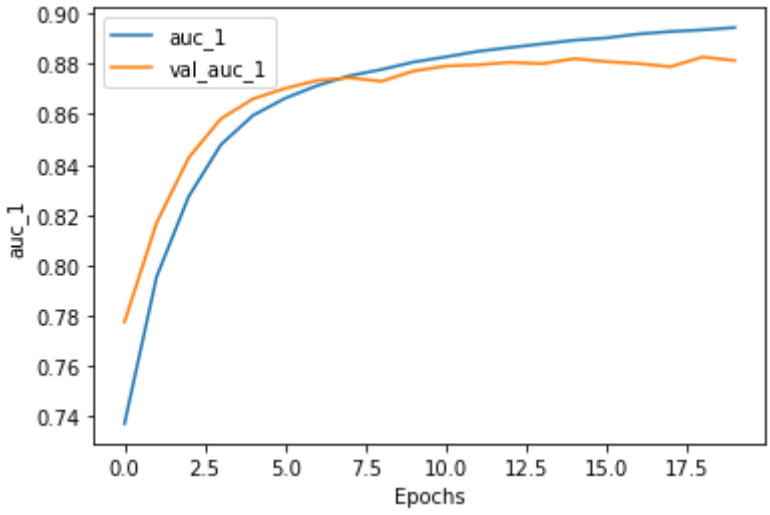
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 80, 16)            1600000

 dropout_1 (Dropout)         (None, 80, 16)            0

 global_average_pooling1d_2   (None, 16)               0
 (GlobalAveragePooling1D)

 dense_4 (Dense)             (None, 128)               2176

 dense_5 (Dense)             (None, 28)                3612

=================================================================
Total params: 1,605,788
Trainable params: 1,605,788
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
6250/6250 - 23s - loss: 0.1604 - auc_1: 0.7370 - val_loss: 0.1491 - val_auc_1: 0.777
4 - 23s/epoch - 4ms/step
Epoch 2/20
6250/6250 - 21s - loss: 0.1458 - auc_1: 0.7954 - val_loss: 0.1401 - val_auc_1: 0.816
9 - 21s/epoch - 3ms/step
```

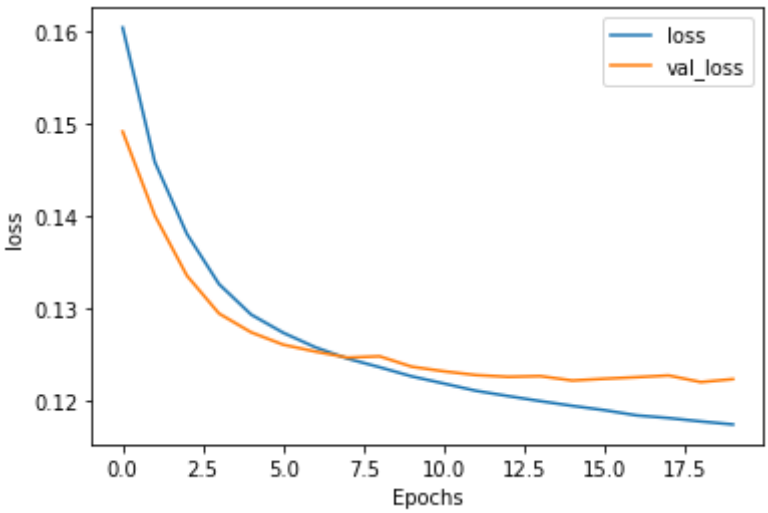
```
Epoch 3/20
6250/6250 - 21s - loss: 0.1381 - auc_1: 0.8273 - val_loss: 0.1335 - val_auc_1: 0.842
7 - 21s/epoch - 3ms/step
Epoch 4/20
6250/6250 - 20s - loss: 0.1326 - auc_1: 0.8478 - val_loss: 0.1294 - val_auc_1: 0.858
2 - 20s/epoch - 3ms/step
Epoch 5/20
6250/6250 - 20s - loss: 0.1293 - auc_1: 0.8595 - val_loss: 0.1274 - val_auc_1: 0.866
1 - 20s/epoch - 3ms/step
Epoch 6/20
6250/6250 - 21s - loss: 0.1273 - auc_1: 0.8663 - val_loss: 0.1260 - val_auc_1: 0.870
2 - 21s/epoch - 3ms/step
Epoch 7/20
6250/6250 - 22s - loss: 0.1258 - auc_1: 0.8714 - val_loss: 0.1253 - val_auc_1: 0.873
4 - 22s/epoch - 3ms/step
Epoch 8/20
6250/6250 - 21s - loss: 0.1245 - auc_1: 0.8752 - val_loss: 0.1247 - val_auc_1: 0.874
5 - 21s/epoch - 3ms/step
Epoch 9/20
6250/6250 - 20s - loss: 0.1236 - auc_1: 0.8777 - val_loss: 0.1248 - val_auc_1: 0.873
1 - 20s/epoch - 3ms/step
Epoch 10/20
6250/6250 - 20s - loss: 0.1226 - auc_1: 0.8807 - val_loss: 0.1237 - val_auc_1: 0.877
2 - 20s/epoch - 3ms/step
Epoch 11/20
6250/6250 - 21s - loss: 0.1218 - auc_1: 0.8828 - val_loss: 0.1232 - val_auc_1: 0.879
1 - 21s/epoch - 3ms/step
Epoch 12/20
6250/6250 - 20s - loss: 0.1211 - auc_1: 0.8850 - val_loss: 0.1228 - val_auc_1: 0.879
6 - 20s/epoch - 3ms/step
Epoch 13/20
6250/6250 - 20s - loss: 0.1205 - auc_1: 0.8865 - val_loss: 0.1226 - val_auc_1: 0.880
6 - 20s/epoch - 3ms/step
Epoch 14/20
6250/6250 - 20s - loss: 0.1199 - auc_1: 0.8879 - val_loss: 0.1226 - val_auc_1: 0.880
1 - 20s/epoch - 3ms/step
Epoch 15/20
6250/6250 - 21s - loss: 0.1194 - auc_1: 0.8894 - val_loss: 0.1222 - val_auc_1: 0.882
0 - 21s/epoch - 3ms/step
Epoch 16/20
6250/6250 - 21s - loss: 0.1190 - auc_1: 0.8903 - val_loss: 0.1224 - val_auc_1: 0.880
9 - 21s/epoch - 3ms/step
Epoch 17/20
6250/6250 - 20s - loss: 0.1184 - auc_1: 0.8919 - val_loss: 0.1225 - val_auc_1: 0.880
1 - 20s/epoch - 3ms/step
Epoch 18/20
6250/6250 - 21s - loss: 0.1181 - auc_1: 0.8928 - val_loss: 0.1227 - val_auc_1: 0.878
9 - 21s/epoch - 3ms/step
Epoch 19/20
6250/6250 - 21s - loss: 0.1177 - auc_1: 0.8935 - val_loss: 0.1220 - val_auc_1: 0.882
7 - 21s/epoch - 3ms/step
Epoch 20/20
6250/6250 - 21s - loss: 0.1174 - auc_1: 0.8944 - val_loss: 0.1223 - val_auc_1: 0.881
3 - 21s/epoch - 3ms/step
```

In [15]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
```

```
  plt.show()

plot_graphs(history, "auc_1")
plot_graphs(history, "loss")
```





Other model evaluated less conclusive

MODEL 2

In [ ]:

```
from keras.layers import LSTM
from keras.layers import Bidirectional

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(LSTM(8, return_sequences=True)),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(28 , activation='sigmoid')  # change its shape to the maxi
])

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=[tf.keras.metrics.

#model.summary()
```

loss='categorical_crossentropy' : auc_2: 0.5032

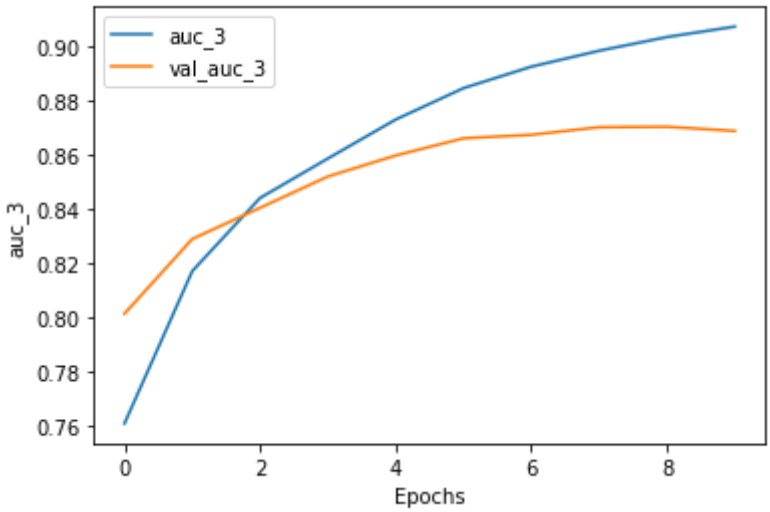
In [ ]:
```
# keras.utils.plot_model(model, "27 sentiments analysis.png", show_shapes=True)
```

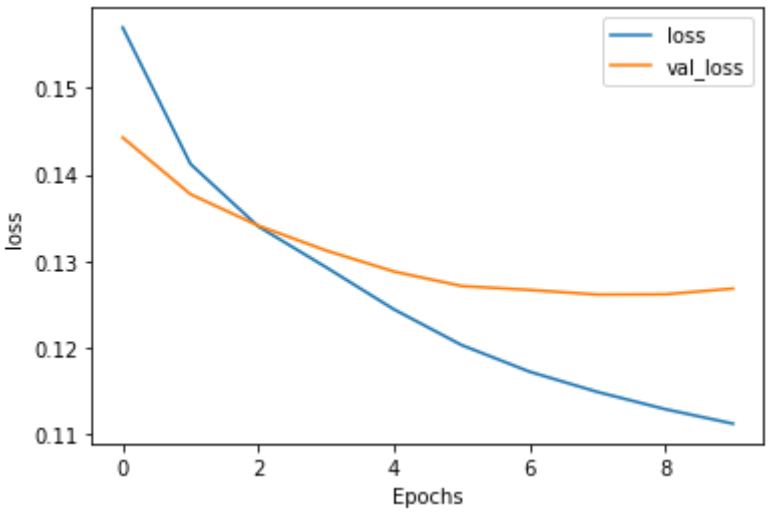
In [ ]:
```
from keras.callbacks import LearningRateScheduler

# # we add learning rate scheduler to change because we are on a plateau
# # for that we add it in callback

# callbacks = [
#     ReduceLROnPlateau(), # monitors a quantity and if no improvement is seen for a
#     ModelCheckpoint(filepath='model_sparse', save_best_only=True)
# ]

##############################
def lr_scheduler(epoch, lr):
    decay_rate = 0.1
    decay_step = 90
    if epoch % decay_step == 0 and epoch:
        return lr * decay_rate
    return lr
callbacks = [
    keras.callbacks.LearningRateScheduler(lr_scheduler, verbose=1)
]
################################
#callbacks = keras.callbacks.ModelCheckpoint("model_sparse.h5", save_best_only=True)
```

In [ ]:
```
num_epochs = 10
history = model.fit(training_padded, training_labels, epochs=num_epochs, validation_
```

```
Epoch 1/10
6250/6250 - 60s - loss: 0.1570 - auc_3: 0.7611 - val_loss: 0.1443 - val_auc_3: 0.801
5 - 60s/epoch - 10ms/step
Epoch 2/10
6250/6250 - 57s - loss: 0.1412 - auc_3: 0.8171 - val_loss: 0.1377 - val_auc_3: 0.828
9 - 57s/epoch - 9ms/step
Epoch 3/10
6250/6250 - 57s - loss: 0.1340 - auc_3: 0.8440 - val_loss: 0.1341 - val_auc_3: 0.840
4 - 57s/epoch - 9ms/step
Epoch 4/10
6250/6250 - 56s - loss: 0.1293 - auc_3: 0.8585 - val_loss: 0.1312 - val_auc_3: 0.851
9 - 56s/epoch - 9ms/step
Epoch 5/10
6250/6250 - 56s - loss: 0.1244 - auc_3: 0.8729 - val_loss: 0.1288 - val_auc_3: 0.859
6 - 56s/epoch - 9ms/step
Epoch 6/10
6250/6250 - 57s - loss: 0.1203 - auc_3: 0.8844 - val_loss: 0.1271 - val_auc_3: 0.866
0 - 57s/epoch - 9ms/step
Epoch 7/10
6250/6250 - 56s - loss: 0.1172 - auc_3: 0.8923 - val_loss: 0.1267 - val_auc_3: 0.867
2 - 56s/epoch - 9ms/step
Epoch 8/10
6250/6250 - 56s - loss: 0.1149 - auc_3: 0.8982 - val_loss: 0.1261 - val_auc_3: 0.870
0 - 56s/epoch - 9ms/step
Epoch 9/10
6250/6250 - 56s - loss: 0.1129 - auc_3: 0.9032 - val_loss: 0.1262 - val_auc_3: 0.870
2 - 56s/epoch - 9ms/step
Epoch 10/10
6250/6250 - 56s - loss: 0.1112 - auc_3: 0.9070 - val_loss: 0.1268 - val_auc_3: 0.868
6 - 56s/epoch - 9ms/step
```

In [ ]:
```
def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "auc_3")
plot_graphs(history, "loss")
```





MODEL 3

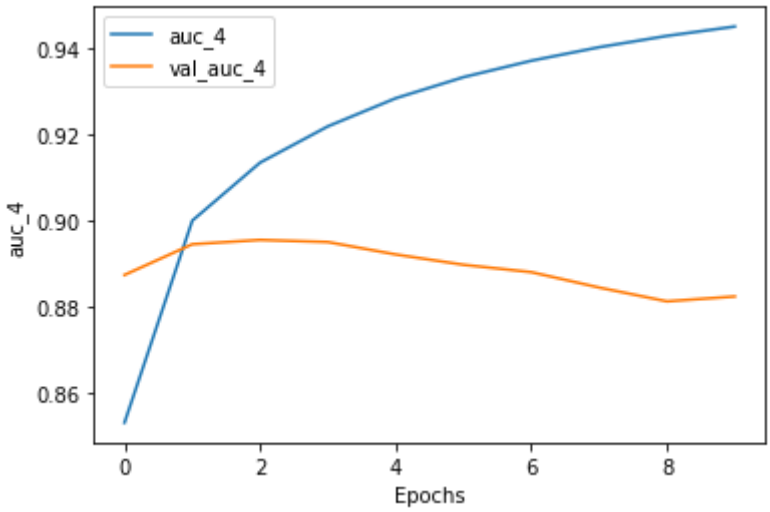
In [ ]:
```
#  Here the length refers to the number of rows of the filter,
# here it is the dimension of the entire word embedding or the entire character repr
filter_length = 300


model = tf.keras.Sequential()
model.add(Embedding(vocab_size, 20, input_length=max_length))
#model.add(Dropout(0.5))
model.add(Conv1D(filter_length, 3, padding='valid', activation='relu', strides=1))
model.add(GlobalMaxPool1D())
model.add(Dense(training_labels.shape[1]))
model.add(Activation('sigmoid'))

#model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

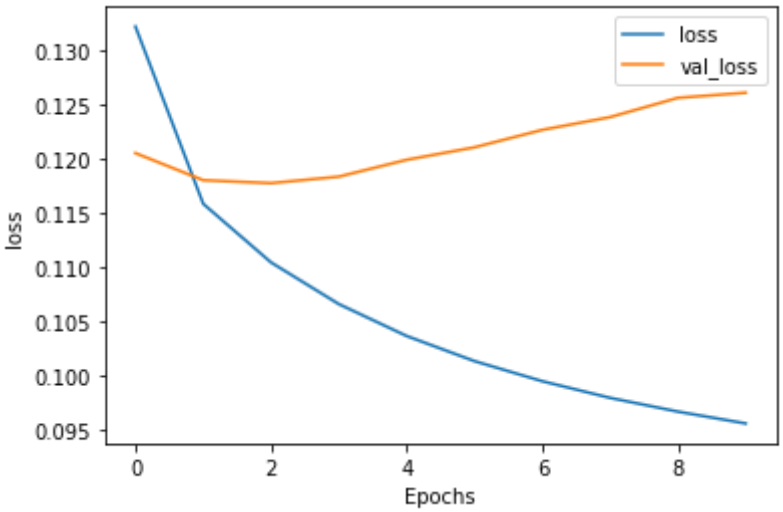
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[tf.keras.metric

callbacks = [
    ReduceLROnPlateau(),
    ModelCheckpoint(filepath='model-conv1d.h5', save_best_only=True)
]


num_epochs= 10
history = model.fit(training_padded, training_labels,
                    epochs=num_epochs, callbacks=callbacks, validation_data=(testing
```

```
Epoch 1/10
6250/6250 - 26s - loss: 0.1321 - auc_4: 0.8532 - val_loss: 0.1205 - val_auc_4: 0.887
4 - lr: 0.0010 - 26s/epoch - 4ms/step
Epoch 2/10
6250/6250 - 22s - loss: 0.1158 - auc_4: 0.9000 - val_loss: 0.1180 - val_auc_4: 0.894
5 - lr: 0.0010 - 22s/epoch - 3ms/step
Epoch 3/10
6250/6250 - 22s - loss: 0.1105 - auc_4: 0.9134 - val_loss: 0.1177 - val_auc_4: 0.895
5 - lr: 0.0010 - 22s/epoch - 3ms/step
Epoch 4/10
6250/6250 - 23s - loss: 0.1066 - auc_4: 0.9218 - val_loss: 0.1183 - val_auc_4: 0.895
0 - lr: 0.0010 - 23s/epoch - 4ms/step
Epoch 5/10
6250/6250 - 21s - loss: 0.1037 - auc_4: 0.9284 - val_loss: 0.1199 - val_auc_4: 0.892
2 - lr: 0.0010 - 21s/epoch - 3ms/step
Epoch 6/10
6250/6250 - 21s - loss: 0.1014 - auc_4: 0.9332 - val_loss: 0.1210 - val_auc_4: 0.889
8 - lr: 0.0010 - 21s/epoch - 3ms/step
Epoch 7/10
6250/6250 - 21s - loss: 0.0995 - auc_4: 0.9370 - val_loss: 0.1226 - val_auc_4: 0.888
1 - lr: 0.0010 - 21s/epoch - 3ms/step
Epoch 8/10
6250/6250 - 21s - loss: 0.0980 - auc_4: 0.9402 - val_loss: 0.1238 - val_auc_4: 0.884
5 - lr: 0.0010 - 21s/epoch - 3ms/step
Epoch 9/10
6250/6250 - 20s - loss: 0.0967 - auc_4: 0.9428 - val_loss: 0.1256 - val_auc_4: 0.881
3 - lr: 0.0010 - 20s/epoch - 3ms/step
Epoch 10/10
6250/6250 - 21s - loss: 0.0956 - auc_4: 0.9450 - val_loss: 0.1261 - val_auc_4: 0.882
4 - lr: 0.0010 - 21s/epoch - 3ms/step
```

In [ ]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "auc_4")
plot_graphs(history, "loss")
```





MODEL 3

sans callback

avec dropout

https://stackabuse.com/python-for-nlp-multi-label-text-classification-with-keras/

https://www.kaggle.com/code/bansodesandeep/multilabel-cuisine-classification-cnn-dnn-lstm

In [ ]:

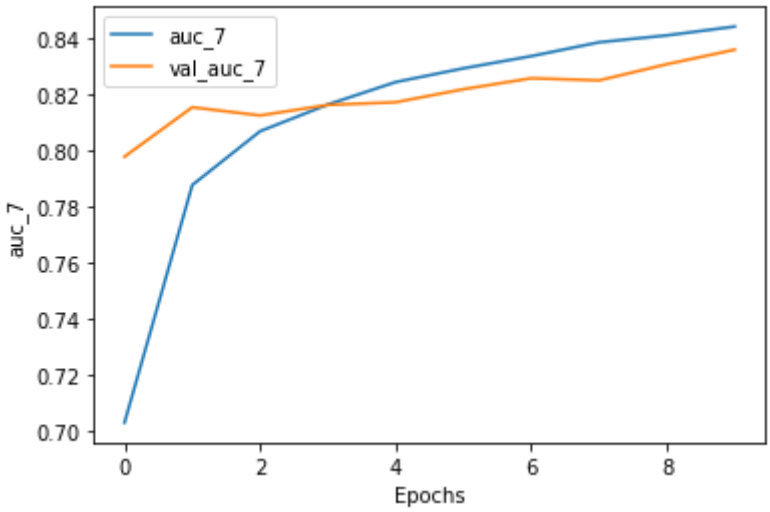
```
#  Here the length refers to the number of rows of the filter,
# here it is the dimension of the entire word embedding or the entire character repr
filter_length = 300


model = tf.keras.Sequential()
model.add(Embedding(vocab_size, 20, input_length=max_length))
model.add(Dropout(0.5))
model.add(Conv1D(filter_length, 3, padding='valid', activation='relu', strides=1))
model.add(GlobalMaxPool1D())
model.add(Dense(training_labels.shape[1]))
model.add(Activation('sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[tf.keras.metric
```

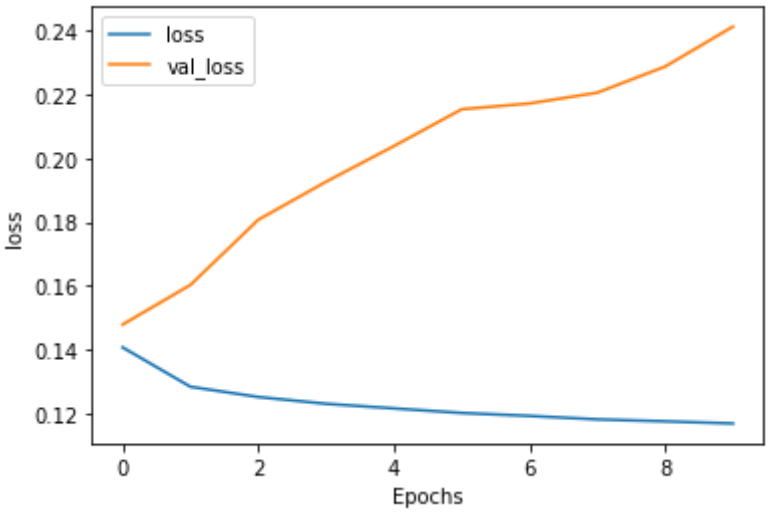
```
callbacks = [
    ReduceLROnPlateau(),
    ModelCheckpoint(filepath='model-conv1d.h5', save_best_only=True)
]


num_epochs= 10
history = model.fit(training_padded, training_labels,
                    epochs=num_epochs, validation_data=(testing_padded, testing_labe
```

```
Epoch 1/10
6250/6250 - 25s - loss: 0.1407 - auc_7: 0.7028 - val_loss: 0.1479 - val_auc_7: 0.797
9 - 25s/epoch - 4ms/step
Epoch 2/10
6250/6250 - 23s - loss: 0.1284 - auc_7: 0.7878 - val_loss: 0.1603 - val_auc_7: 0.815
6 - 23s/epoch - 4ms/step
Epoch 3/10
6250/6250 - 24s - loss: 0.1252 - auc_7: 0.8070 - val_loss: 0.1807 - val_auc_7: 0.812
7 - 24s/epoch - 4ms/step
Epoch 4/10
6250/6250 - 26s - loss: 0.1231 - auc_7: 0.8166 - val_loss: 0.1927 - val_auc_7: 0.816
4 - 26s/epoch - 4ms/step
Epoch 5/10
6250/6250 - 23s - loss: 0.1216 - auc_7: 0.8246 - val_loss: 0.2038 - val_auc_7: 0.817
4 - 23s/epoch - 4ms/step
Epoch 6/10
6250/6250 - 24s - loss: 0.1202 - auc_7: 0.8295 - val_loss: 0.2154 - val_auc_7: 0.822
1 - 24s/epoch - 4ms/step
Epoch 7/10
6250/6250 - 23s - loss: 0.1193 - auc_7: 0.8339 - val_loss: 0.2172 - val_auc_7: 0.826
0 - 23s/epoch - 4ms/step
Epoch 8/10
6250/6250 - 23s - loss: 0.1182 - auc_7: 0.8388 - val_loss: 0.2206 - val_auc_7: 0.825
2 - 23s/epoch - 4ms/step
Epoch 9/10
6250/6250 - 23s - loss: 0.1176 - auc_7: 0.8413 - val_loss: 0.2287 - val_auc_7: 0.831
0 - 23s/epoch - 4ms/step
Epoch 10/10
6250/6250 - 23s - loss: 0.1169 - auc_7: 0.8444 - val_loss: 0.2413 - val_auc_7: 0.836
2 - 23s/epoch - 4ms/step
```

In [ ]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "auc_7")
plot_graphs(history, "loss")
```





MODEL 3

sans callback

avec dropout

sans filter length, kernel size null

In [ ]:

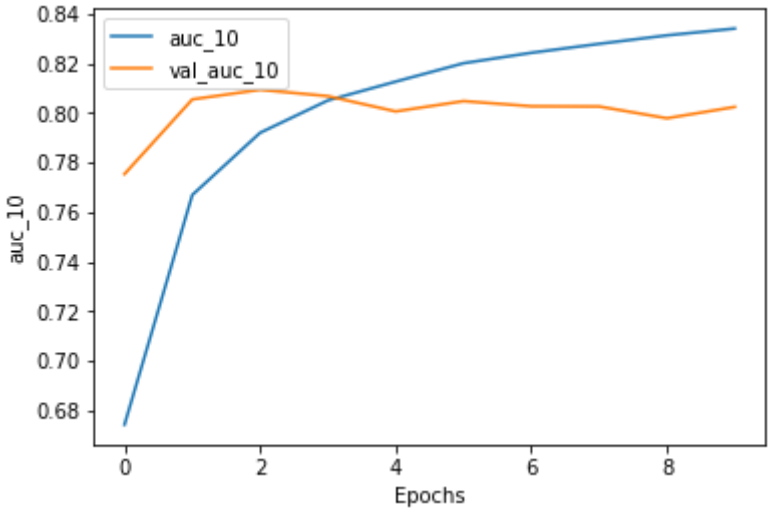
```
#  Here the length refers to the number of rows of the filter,
# here it is the dimension of the entire word embedding or the entire character repr
filter_length = 100


model = tf.keras.Sequential()
model.add(Embedding(vocab_size, 20, input_length=max_length))
model.add(Dropout(0.5))
model.add(Conv1D(filter_length, 3, padding='valid', activation='relu', strides=1))
model.add(GlobalMaxPool1D())
model.add(Dense(training_labels.shape[1]))
model.add(Activation('sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[tf.keras.metric
```

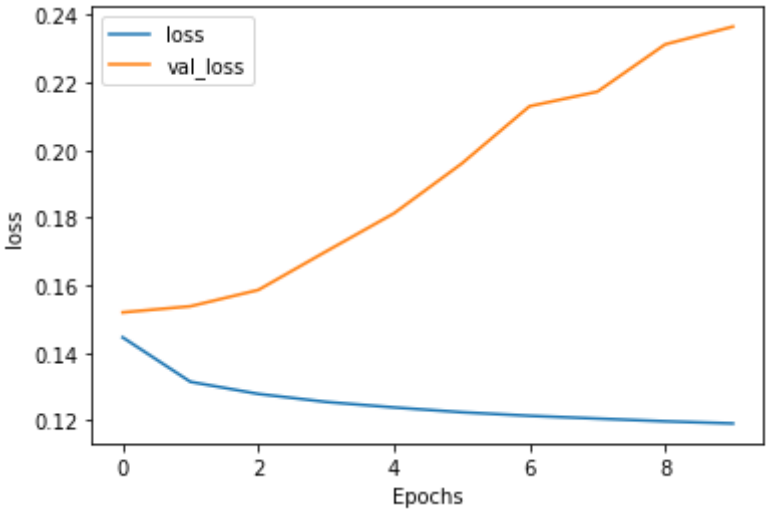
```
callbacks = [
    ReduceLROnPlateau(),
    ModelCheckpoint(filepath='model-conv1d.h5', save_best_only=True)
]


num_epochs= 10
history = model.fit(training_padded, training_labels,
                    epochs=num_epochs, validation_data=(testing_padded, testing_labe
```

```
Epoch 1/10
6250/6250 - 26s - loss: 0.1445 - auc_10: 0.6742 - val_loss: 0.1519 - val_auc_10: 0.7
753 - 26s/epoch - 4ms/step
Epoch 2/10
6250/6250 - 23s - loss: 0.1314 - auc_10: 0.7669 - val_loss: 0.1537 - val_auc_10: 0.8
055 - 23s/epoch - 4ms/step
Epoch 3/10
6250/6250 - 24s - loss: 0.1278 - auc_10: 0.7920 - val_loss: 0.1585 - val_auc_10: 0.8
093 - 24s/epoch - 4ms/step
Epoch 4/10
6250/6250 - 23s - loss: 0.1254 - auc_10: 0.8050 - val_loss: 0.1699 - val_auc_10: 0.8
068 - 23s/epoch - 4ms/step
Epoch 5/10
6250/6250 - 23s - loss: 0.1238 - auc_10: 0.8126 - val_loss: 0.1812 - val_auc_10: 0.8
006 - 23s/epoch - 4ms/step
Epoch 6/10
6250/6250 - 23s - loss: 0.1224 - auc_10: 0.8200 - val_loss: 0.1959 - val_auc_10: 0.8
048 - 23s/epoch - 4ms/step
Epoch 7/10
6250/6250 - 23s - loss: 0.1213 - auc_10: 0.8243 - val_loss: 0.2129 - val_auc_10: 0.8
027 - 23s/epoch - 4ms/step
Epoch 8/10
6250/6250 - 23s - loss: 0.1205 - auc_10: 0.8279 - val_loss: 0.2172 - val_auc_10: 0.8
026 - 23s/epoch - 4ms/step
Epoch 9/10
6250/6250 - 23s - loss: 0.1197 - auc_10: 0.8313 - val_loss: 0.2312 - val_auc_10: 0.7
978 - 23s/epoch - 4ms/step
Epoch 10/10
6250/6250 - 23s - loss: 0.1190 - auc_10: 0.8340 - val_loss: 0.2364 - val_auc_10: 0.8
024 - 23s/epoch - 4ms/step
```

In [ ]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "auc_10")
plot_graphs(history, "loss")
```


auc_10
val_auc_10
auc_10
Epochs
0.84
0.82
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0
2
4
6
8


loss
val_loss
loss
Epochs
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0
2
4
6
8