Lab 9 - BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Modelos Generativos

Prof. Eduardo e Prof. Pedro

Objetivos:

- Análise de sarcamos e sentimento com Modelos Sequenciais.

Data da entrega : 21/10

- Complete o código (marcado com ToDo) e quando requisitado, escreva textos diretamente nos notebooks. Onde tiver *None*, substitua pelo seu código.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-Lab7b.pdf"
- Envie o PDF via google FORM

Este notebook é baseado em tensorflow e Keras.

Parte I - Análise de sarcamo com um base de dados em inglês (kaggle) (40pt)

ToDO: Veja o exemplo apresentado durante a aula, sobre detecção de sarcasmo em textos inglês. Execute o notebook do link. Modifique o modelo para um modelo sequencial (redes recorrentes ou um Transformer) e re-faça os testes. Com qual arquitetura você conseguiu melhorar os resultados? Tente explicar o por que.

```
import json
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


vocab_size = 10000
embedding_dim = 16
max_length = 100
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
```

```
training_size = 20000
```

```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sarcasm.json \
    -O /tmp/sarcasm.json
```

```
--2022-09-29 23:34:48--  https://storage.googleapis.com/laurencemoroney-blog.appspot
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.13.240, 172.217
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.13.240|:443...
HTTP request sent, awaiting response... 200 OK
Length: 5643545 (5.4M) [application/json]
Saving to: ‘/tmp/sarcasm.json’

/tmp/sarcasm.json   100%[===================>]   5.38M  --.-KB/s    in 0.02s

2022-09-29 23:34:48 (225 MB/s) - ‘/tmp/sarcasm.json’ saved [5643545/5643545]
```

```
with open("/tmp/sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []

for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])


training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]


tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

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

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


# Need this block to get it to work with TensorFlow 2.x
import numpy as np
training_padded = np.array(training_padded)
training_labels = np.array(training_labels)
```

```
testing_padded = np.array(testing_padded)
testing_labels = np.array(testing_labels)
```

```
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()
```

Modelos

Modelo 1

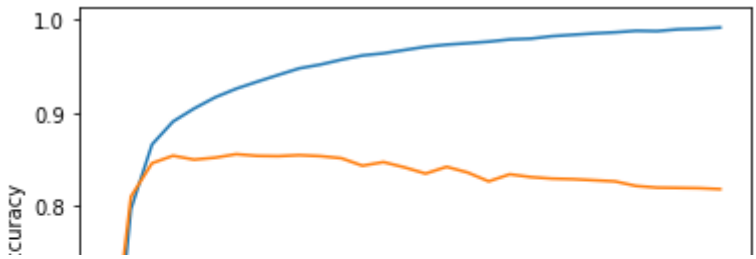
```
model1 = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model1.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
num_epochs = 30
history1 = model1.fit(training_padded, training_labels, epochs=num_epochs, validation_data
```

```
625/625 - 2s - loss: 0.6732 - accuracy: 0.5660 - val_loss: 0.6253 - val_accuracy: 0
Epoch 2/30
625/625 - 1s - loss: 0.4963 - accuracy: 0.7965 - val_loss: 0.4252 - val_accuracy: 0
Epoch 3/30
625/625 - 1s - loss: 0.3437 - accuracy: 0.8659 - val_loss: 0.3671 - val_accuracy: 0
Epoch 4/30
625/625 - 1s - loss: 0.2859 - accuracy: 0.8906 - val_loss: 0.3488 - val_accuracy: 0
Epoch 5/30
625/625 - 1s - loss: 0.2481 - accuracy: 0.9046 - val_loss: 0.3529 - val_accuracy: 0
Epoch 6/30
625/625 - 1s - loss: 0.2193 - accuracy: 0.9166 - val_loss: 0.3466 - val_accuracy: 0
Epoch 7/30
625/625 - 1s - loss: 0.1968 - accuracy: 0.9259 - val_loss: 0.3492 - val_accuracy: 0
Epoch 8/30
625/625 - 1s - loss: 0.1770 - accuracy: 0.9334 - val_loss: 0.3585 - val_accuracy: 0
Epoch 9/30
625/625 - 1s - loss: 0.1606 - accuracy: 0.9407 - val_loss: 0.3702 - val_accuracy: 0
Epoch 10/30
625/625 - 1s - loss: 0.1468 - accuracy: 0.9478 - val_loss: 0.3826 - val_accuracy: 0
Epoch 11/30
625/625 - 1s - loss: 0.1339 - accuracy: 0.9518 - val_loss: 0.3994 - val_accuracy: 0
Epoch 12/30
625/625 - 1s - loss: 0.1253 - accuracy: 0.9571 - val_loss: 0.4180 - val_accuracy: 0
Epoch 13/30

625/625 - 1s - loss: 0.1135 - accuracy: 0.9617 - val_loss: 0.4505 - val_accuracy: 0
Epoch 14/30
```

```
625/625 - 1s - loss: 0.1055 - accuracy: 0.9639 - val_loss: 0.4540 - val_accuracy: 0
Epoch 15/30
625/625 - 1s - loss: 0.0982 - accuracy: 0.9675 - val_loss: 0.4861 - val_accuracy: 0
Epoch 16/30
625/625 - 1s - loss: 0.0894 - accuracy: 0.9710 - val_loss: 0.5269 - val_accuracy: 0
Epoch 17/30
625/625 - 1s - loss: 0.0828 - accuracy: 0.9732 - val_loss: 0.5242 - val_accuracy: 0
Epoch 18/30
625/625 - 1s - loss: 0.0782 - accuracy: 0.9748 - val_loss: 0.5566 - val_accuracy: 0
Epoch 19/30
625/625 - 1s - loss: 0.0717 - accuracy: 0.9765 - val_loss: 0.6249 - val_accuracy: 0
Epoch 20/30
625/625 - 2s - loss: 0.0668 - accuracy: 0.9789 - val_loss: 0.6062 - val_accuracy: 0
Epoch 21/30
625/625 - 1s - loss: 0.0628 - accuracy: 0.9797 - val_loss: 0.6277 - val_accuracy: 0
Epoch 22/30
625/625 - 1s - loss: 0.0574 - accuracy: 0.9823 - val_loss: 0.6724 - val_accuracy: 0
Epoch 23/30
625/625 - 1s - loss: 0.0531 - accuracy: 0.9839 - val_loss: 0.6947 - val_accuracy: 0
Epoch 24/30
625/625 - 1s - loss: 0.0497 - accuracy: 0.9854 - val_loss: 0.7222 - val_accuracy: 0
Epoch 25/30
625/625 - 1s - loss: 0.0455 - accuracy: 0.9865 - val_loss: 0.7485 - val_accuracy: 0
Epoch 26/30
625/625 - 2s - loss: 0.0416 - accuracy: 0.9882 - val_loss: 0.8258 - val_accuracy: 0
Epoch 27/30
625/625 - 2s - loss: 0.0398 - accuracy: 0.9878 - val_loss: 0.8395 - val_accuracy: 0
Epoch 28/30
625/625 - 1s - loss: 0.0369 - accuracy: 0.9898 - val_loss: 0.8738 - val_accuracy: 0
Epoch 29/30
625/625 - 1s - loss: 0.0350 - accuracy: 0.9904 - val_loss: 0.8911 - val_accuracy: 0
```

```
plot_graphs(history1, "accuracy")
plot_graphs(history1, "loss")
```



Modelo 2

```
training_padded.shape
```

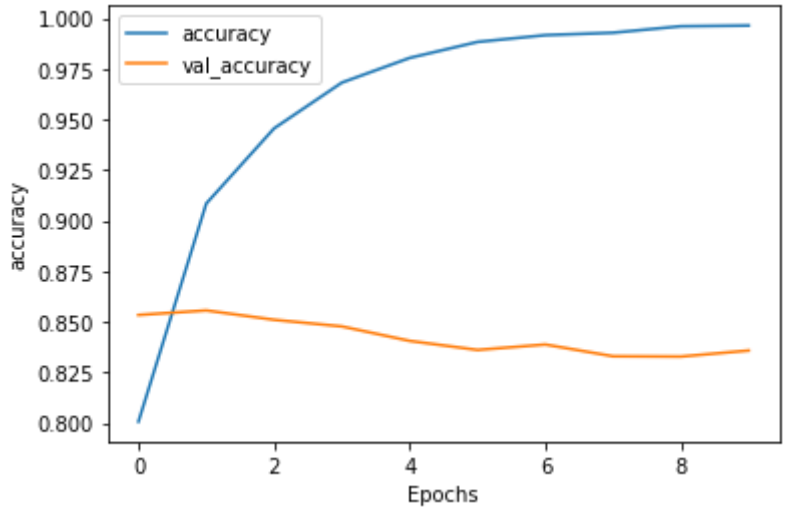
```
(20000, 100)
```

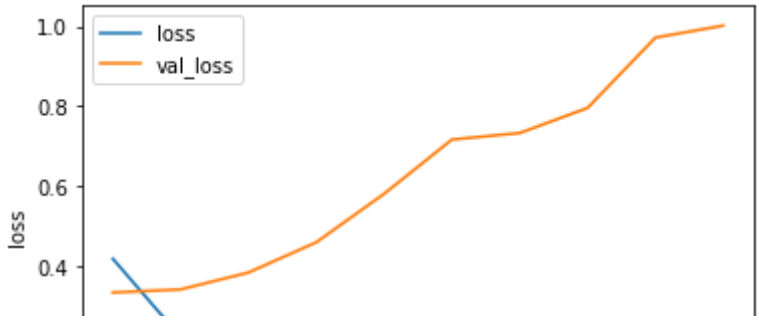
```
model2 = Sequential()
model2.add(tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length))
model2.add(tf.keras.layers.Bidirectional(tf.keras.layers.GRU(24)))
model2.add(tf.keras.layers.Dense(8, activation='relu')),
model2.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model2.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
num_epochs = 10
history2 = model2.fit(training_padded, training_labels, epochs=num_epochs, validation_data
```

```
Epoch 1/10
625/625 - 20s - loss: 0.4197 - accuracy: 0.8006 - val_loss: 0.3354 - val_accuracy: 0
Epoch 2/10
625/625 - 15s - loss: 0.2239 - accuracy: 0.9084 - val_loss: 0.3432 - val_accuracy: 0
Epoch 3/10
625/625 - 15s - loss: 0.1445 - accuracy: 0.9456 - val_loss: 0.3853 - val_accuracy: 0
Epoch 4/10
625/625 - 16s - loss: 0.0911 - accuracy: 0.9682 - val_loss: 0.4608 - val_accuracy: 0
Epoch 5/10
625/625 - 16s - loss: 0.0598 - accuracy: 0.9804 - val_loss: 0.5818 - val_accuracy: 0
Epoch 6/10
625/625 - 15s - loss: 0.0380 - accuracy: 0.9883 - val_loss: 0.7171 - val_accuracy: 0
Epoch 7/10
625/625 - 15s - loss: 0.0277 - accuracy: 0.9916 - val_loss: 0.7333 - val_accuracy: 0
Epoch 8/10
625/625 - 16s - loss: 0.0227 - accuracy: 0.9929 - val_loss: 0.7960 - val_accuracy: 0
Epoch 9/10
625/625 - 15s - loss: 0.0134 - accuracy: 0.9961 - val_loss: 0.9714 - val_accuracy: 0
Epoch 10/10
625/625 - 15s - loss: 0.0113 - accuracy: 0.9964 - val_loss: 1.0013 - val_accuracy: 0
```

```
plot_graphs(history2, "accuracy")
plot_graphs(history2, "loss")
```





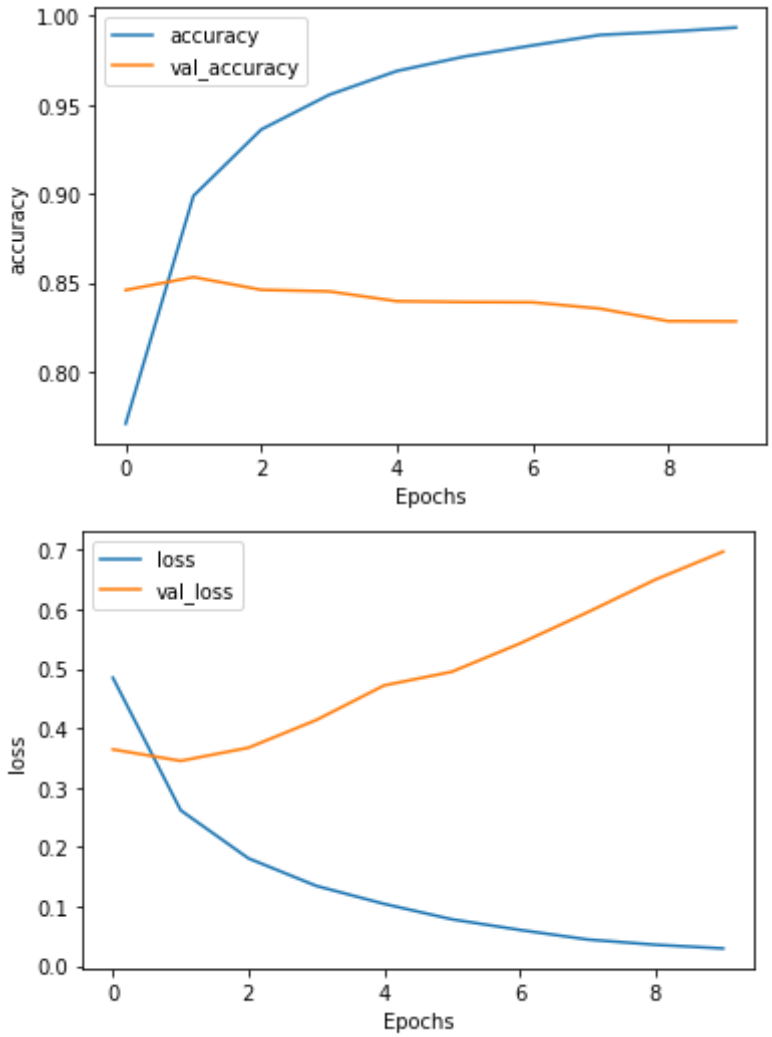
Modelo 3

```
model3 = Sequential()
model3.add(tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length))
model3.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(4)))
model3.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model3.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
num_epochs = 10
history3 = model3.fit(training_padded, training_labels, epochs=num_epochs, validation_data
```

```
Epoch 1/10
625/625 - 16s - loss: 0.4850 - accuracy: 0.7709 - val_loss: 0.3643 - val_accuracy: 0
Epoch 2/10
625/625 - 13s - loss: 0.2617 - accuracy: 0.8988 - val_loss: 0.3450 - val_accuracy: 0
Epoch 3/10
625/625 - 13s - loss: 0.1808 - accuracy: 0.9361 - val_loss: 0.3671 - val_accuracy: 0
Epoch 4/10
625/625 - 13s - loss: 0.1345 - accuracy: 0.9556 - val_loss: 0.4138 - val_accuracy: 0
Epoch 5/10
625/625 - 13s - loss: 0.1042 - accuracy: 0.9689 - val_loss: 0.4721 - val_accuracy: 0
Epoch 6/10
625/625 - 13s - loss: 0.0782 - accuracy: 0.9771 - val_loss: 0.4951 - val_accuracy: 0
Epoch 7/10
625/625 - 12s - loss: 0.0603 - accuracy: 0.9833 - val_loss: 0.5426 - val_accuracy: 0
Epoch 8/10
625/625 - 12s - loss: 0.0441 - accuracy: 0.9892 - val_loss: 0.5953 - val_accuracy: 0
Epoch 9/10
625/625 - 13s - loss: 0.0353 - accuracy: 0.9911 - val_loss: 0.6503 - val_accuracy: 0
Epoch 10/10
625/625 - 13s - loss: 0.0291 - accuracy: 0.9934 - val_loss: 0.6974 - val_accuracy: 0
```

```
plot_graphs(history3, "accuracy")
plot_graphs(history3, "loss")
```



O Modelo 2 com uma Rede Neural Recorrente Bidirecional(GRU) teve uma melhor acurácia tanto no treino e na validação, e melhor do que a rede neural comum, pois a Rede Recorrente implementada considera a ordem das palavras, e tem uma certa memória, o que é de suma importância na avaliação de um texto.

Parte II - Análise de sentimento de um base em português (60pt)

A seguir, faremos um teste semelhante com dados em português. Os dados abaixo foram retirados do Twitter, em 2017. As contas monitoradas estavam comentando sobre o governo de Minas Gerais, e foi levantado o sentimento com relação ao governo.

Carregando os pacotes

```
import numpy as np
import pandas as pd
import re
import string
import tensorflow as tf
import tensorflow_datasets as tfds

from sklearn import metrics
from sklearn.model_selection import train_test_split
from string import punctuation
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, SpatialDropout1D, LSTM, Dense, Flatten
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Carregando a base (arquivo CSV - Tweets_minas_2017.dat)

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
URL = '/content/drive/MyDrive/Tweets_minas_2017.dat'
dataframe = pd.read_csv(URL, encoding='utf-8')
dataframe.head()
```

	Unnamed: 0	Created At	Text	Geo Coordinates.latitude	Coordinates.longit
0	0	Sun Jan 08 01:22:05 +0000 2017	⛪ @ Catedral de Santo Antônio - Governador ...	NaN	N
1	1	Sun Jan 08 01:49:01 +0000 2017	@ Governador Valadares, Minas Gerais https:/...	-41.9333	-18

Pré-processamento dos dados

```
# configura/renomeia o dataframe para campos que façam mais sentido
np.random.seed(42)
raw_df = (dataframe.rename(columns={"Created At": "publication_date",
                                    "Text": "tweet",
                                    "Retweet Count" : "num_retweets",
                                    "Username": "username",
                                    "Classificacao": "sentiment"})
                   .loc[:, ["publication_date", "tweet", "num_retweets",  "username", "sentime
raw_df["publication_date"] = pd.to_datetime(raw_df["publication_date"],infer_datetime_form
raw_df["sentiment"] = raw_df["sentiment"].replace({"Negativo": 0, "Neutro": np.random.choi

raw_df.head()
```

	publication_date	tweet	num_retweets	username	sentiment
0	2017-01-08 01:22:05+00:00	⛪ @ Catedral de Santo Antônio - Governador ...	0	Leonardo C Schneider	0
1	2017-01-08 01:49:01+00:00	@ Governador Valadares, Minas Gerais https:/...	0	Wândell	0
2	2017-01-08 01:01:46+00:00	@ Governador Valadares, Minas Gerais https:...	0	Wândell	0

```
# veja a quantidade de exemplos positivos e negativos
neg, pos = np.bincount(raw_df['sentiment'])
total = neg + pos
print('Exemplos:\n    Total: {} \n    Negativos: {} ({:.2f}% do total)\n    Positivos: {}
    total, neg, 100*neg/total, pos, 100 * pos / total))
```

```
Exemplos:
    Total: 8199
    Negativos: 4899 (59.75% do total)
    Positivos: 3300 (40.25% do total)
```

Pode-se limpar o texto, excluindo-se caracteres estranhos ou infrequentes. Por exemplo, veja a função clean_text.

```
# limpa o texto
def clean_text(text):

    ## Remove puncuation
    text = text.translate(string.punctuation)

    ## coloca tudo em caixa baixa
    text = text.lower().split()

    text = " ".join(text)
    ## limpa/altera
    text = re.sub(r"pra", "para", text)
    text = re.sub(r"eh", "e", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"http://", "", text)
    text = re.sub(r"https://", "", text)
    text = re.sub(r"'", " ", text)
    text = re.sub(r":", " : ", text)
    ##
    text = text.split()

    return text
```

Padronizar o texto ajuda o modelo de aprendizagem de máquina. Aplique a função de limpeza nos dados.

```
# Aplica a função acima e limpa o texto
raw_df['tweet'] = raw_df['tweet'].map(lambda x: clean_text(x))


#  a data da publicação não parece interessante para o nosso contexto. Vamos descarta-la,
cleaned_df = raw_df.copy()

# descara data da publicação, numero de retweets e username
cleaned_df.pop('publication_date')
cleaned_df.pop('num_retweets')
cleaned_df.pop('username')

cleaned_df.head()
```

	tweet	sentiment
0	[���⛪, @, catedral, de, santo, antônio, -, gov...	0
1	[�, @, governador, valadares, minas, gerais, h...	0
2	[��, @, governador, valadares, minas, gerais, ...	0
3	[���, https, :, t, co, bndso34qk0]	0
4	[���, psol, vai, questionar, aumento, de, vere...	0

```
dataframe = cleaned_df.copy()
labels = dataframe.pop('sentiment')
data = dataframe.pop('tweet')
```

Use o objeto Tokenizer para separar o texto em vários tokens. Uma instância de dados deve ter um tamanho máximo para modelos de redes neurais. Consideraremos, aqui, que um tweet terá 30 tokens, no máximo. Tweets menores serão completados com zero (zero padding, pad_sequences). Veja que o objeto Tokenizer também oferece uma forma de se filtrar caracteres não desejados.

```
tk = Tokenizer(lower=True, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
tk.fit_on_texts(data)
word_index = tk.word_index
print(len(word_index))
print(word_index)

max_len = 30 # tamanho maximo para um twitte
train_tokenized = tk.texts_to_sequences(data)
X = np.array(pad_sequences(train_tokenized, maxlen=max_len))

y = np.array(labels)

print(X[0])
```

```
13807
{':': 1, 'de': 2, 'https': 3, 't': 4, 'co': 5, 'em': 6, 'minas': 7, 'rt': 8, '-': 9,
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0 4554  946 4555    2 2119 4556    9   33   76   11    3    1    4
    5 4557]
```

```
print(X.shape)
```

```
(8199, 30)
```

Observe que cada token recebe um valor único. O comando abaixo imprime a instância X[100].

```
print(X[100])
```

```
[   0    0    0    0    0    0    0    0    0 4809 1645  270 4810   35
```

```
       4811   41 2131 4812   35 2557 4813  253 4814    1   99    3    1    4
          5 4815]
```

```
# divide em 80% treino e 20% validação.
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2)
train_X, val_X, train_y, val_y = train_test_split(train_X, train_y, test_size=0.2)
print(len(train_X), 'exemplos de treino')
print(len(val_X), 'exemplos de validação')
print(len(test_X), 'exemplos de test')
```

```
     5247 exemplos de treino
     1312 exemplos de validação
     1640 exemplos de test
```

ToDo: Construa uma rede recorrente para fazer uma análise de sentimentos. (40pt)

```
len(word_index)
```

```
     13807
```

```
X.max()
```

```
     13807
```

```
# Se necessário, mude o shape de train_X e val_X

# Monta o modelo sequencial
embed_dim = 60 # fique a vontade para escolher a quantidade de dimensões da camada de embe
max_fatures = X.max() + 1 # número máximo de palavras da base de treino + uma

model = tf.keras.Sequential([tf.keras.layers.Embedding(max_fatures, embedding_dim, input_l
                             tf.keras.layers.Bidirectional(tf.keras.layers.GRU(24)),
                             tf.keras.layers.Dense(8, activation='relu'),
                             tf.keras.layers.Dense(1, activation='sigmoid')])# ToDo ...


# compila
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy']) # Todo : q


# imprime o modelo
print(model.summary())


# treina

n_epochs = 10 # escolha o numero de epocas

model.fit(train_X, train_y, epochs=n_epochs, validation_data=(val_X,val_y))
```

```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_25 (Embedding)    (None, 30, 16)            220928

 bidirectional_29 (Bidirecti  (None, 48)               6048
 onal)

 dense_102 (Dense)           (None, 8)                 392

 dense_103 (Dense)           (None, 1)                 9

=================================================================
Total params: 227,377
Trainable params: 227,377
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
164/164 [==============================] - 5s 18ms/step - loss: 0.4202 - accuracy: 0
Epoch 2/10
164/164 [==============================] - 2s 15ms/step - loss: 0.0607 - accuracy: 0
Epoch 3/10
164/164 [==============================] - 2s 15ms/step - loss: 0.0202 - accuracy: 0
Epoch 4/10
164/164 [==============================] - 2s 14ms/step - loss: 0.0105 - accuracy: 0
Epoch 5/10
164/164 [==============================] - 3s 17ms/step - loss: 0.0070 - accuracy: 0
Epoch 6/10
164/164 [==============================] - 3s 18ms/step - loss: 0.0070 - accuracy: 0
Epoch 7/10
164/164 [==============================] - 2s 14ms/step - loss: 0.0050 - accuracy: 0
Epoch 8/10
164/164 [==============================] - 2s 14ms/step - loss: 0.0041 - accuracy: 0
Epoch 9/10
164/164 [==============================] - 2s 14ms/step - loss: 0.0025 - accuracy: 0
Epoch 10/10
164/164 [==============================] - 2s 14ms/step - loss: 0.0031 - accuracy: 0
<keras.callbacks.History at 0x7f22398fe7d0>
```

ToDo: Avaliação do modelo (10pt)

ToDo: Faça uma avaliação dos resultados encontrados. Você pode usar métricas como precisão, revocação, F1-score, acurácia, etc.

```
Valores próximos de 0 => sentimento Negativo
Valores próximos de 1 => sentimento Positivo
```

```
#Importações
from sklearn.metrics import classification_report
```

```
test_X.shape
```

```
(1640, 30)
```

```
y_pred.shape
```

```
(1640, 1)
```

```
# ToDo: Seu codigo aqui
#Obter Resultados
y_pred = model.predict(test_X, verbose=1)

y_pred_bool = [0 if x < 0.5 else 1 for x in list(y_pred)]
print(classification_report(test_y.reshape(-1), y_pred_bool))
```

```
52/52 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       978
           1       0.98      0.98      0.98       662

    accuracy                           0.98      1640
   macro avg       0.98      0.98      0.98      1640
weighted avg       0.98      0.98      0.98      1640
```

O Modelo teve 98% de acurácia e acertou bem nas duas classes 99% na classe 0(comentário ruim) e 98% na classe 1(comentário positivo)

ToDo: Quais conclusões você chegou?

Clique duas vezes (ou prima Enter) para editar.

Que Modelo acerta muito bem se um comentário é positivo ou negativo, e além disso pode-se concluir que as Redes Neurais Recorrentes são excelentes para trabalhar com textos.

ToDo: Testando as frases (10pt)

```
# função para avaliar um tweet
def sample_predict(sentence):
  max_len = 30 # numero maximo de palavras em um tweet
  tk_sentence = tk.texts_to_sequences(sentence)
  print(tk_sentence)
  encoded_sample_pred_text = np.array(pad_sequences(tk_sentence, maxlen=max_len))
  predictions = model.predict(encoded_sample_pred_text)
  return (predictions)
```

Teste as frases abaixo e verifique se o seu modelo é coerente.

```
sample_pred_text = {'pimentel governa bem, faz um bom trabalho.'}
predictions = sample_predict(sample_pred_text)
print (predictions)
```

```
     [[36, 3439, 1001, 226, 90, 261, 426]]
     [[0.5099271]]
```

Comentário acima é positivo

```
sample_pred_text = {'O governo de minas esta ruim. o governador anda roubando, o povo quer
predictions = sample_predict(sample_pred_text)
print (predictions)
```

```
     [[15, 14, 2, 7, 326, 1285, 15, 33, 2920, 15, 407, 179, 5451, 84, 14]]
     [[3.4901845e-05]]
```

O comentário acima é negativo.

```
sample_pred_text = {'governo fez melhorias na saúde e educação. É um ótimo governo. adoro
predictions = sample_predict(sample_pred_text)
print (predictions)
```

```
     [[14, 1113, 41, 198, 12, 142, 19, 90, 3004, 14, 8443, 36]]
     [[0.9971152]]
```

O comentário acima é positivo

```
sample_pred_text = {'governo de m**da, só sabe roubar!!'}
predictions = sample_predict(sample_pred_text)
print (predictions)
```

```
     [[14, 2, 106, 35, 55, 1676, 4130]]
     [[0.00020024]]
```

O comentário acima é negativo

ToDo: O que você pode dizer sobre as predições?

Que o meu Modelo está acertando corretamente todas as predições.

Produtos pagos do Colab - Cancele os contratos aqui

✓ 0 s concluído à(s) 21:53