DEEPLEARNING

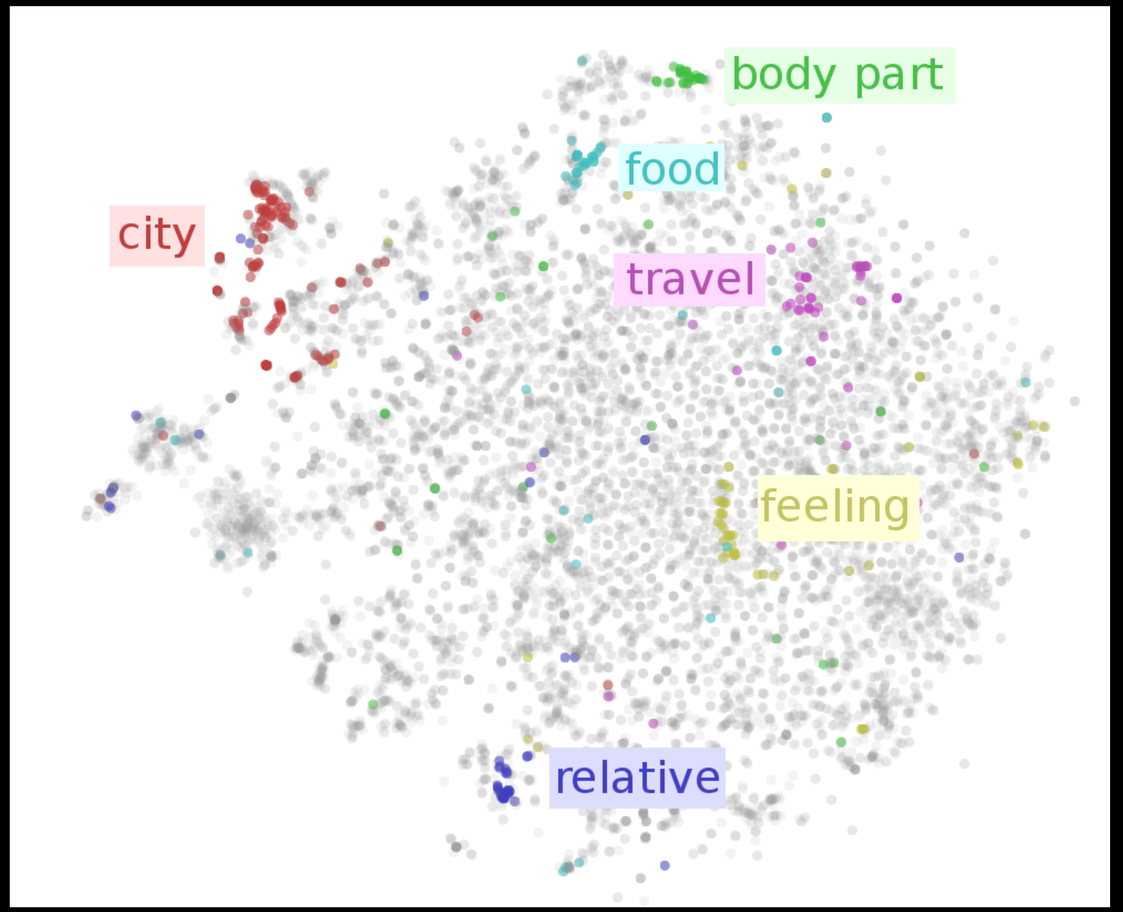
TENSORFLOW

https://www.tensorflow.org

Rafael Vieira Coelho

TENSORFLOW

WORD EMBEDDINGS



A) BASE DE DADOS IMDB

No link abaixo, podemos baixar 50.000 críticas de filmes classificados como positivo ou negativo.

http://ai.stanford.edu/~amaas/data/sentiment/

```
@InProceedings{maas-EtAl:2011:ACL-HLT2011,
  author    = {Maas, Andrew L.  and  Daly, Raymond E.  and  Pham, Peter T.  and  Huang, Dan  and  Ng,
Andrew Y.  and  Potts, Christopher},
  title     = {Learning Word Vectors for Sentiment Analysis},
  booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:
Human Language Technologies},
  month     = {June},
  year      = {2011},
  address   = {Portland, Oregon, USA},
  publisher = {Association for Computational Linguistics},
  pages     = {142--150},
  url       = {http://www.aclweb.org/anthology/P11-1015}
}
```

1) PODEMOS IMPORTAR OS DADOS DESTA FORMA

```
6  import tensorflow_datasets as tfds
7
8  imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```

2) E FORMATAMOS OS DADOS

```
10  import numpy as np
11
12  train_data, test_data = imdb['train'], imdb['test']
13
14  training_sentences = []
15  training_labels = []
16
17  testing_sentences = []
18  testing_labels = []
19
20  # str(s.tonumpy()) is needed in Python3 instead of just s.numpy()
21  for s,l in train_data:
22    training_sentences.append(s.numpy().decode('utf8'))
23    training_labels.append(l.numpy())
24
25  for s,l in test_data:
26    testing_sentences.append(s.numpy().decode('utf8'))
27    testing_labels.append(l.numpy())
28
29  training_labels_final = np.array(training_labels)
30  testing_labels_final = np.array(testing_labels)
```

3) CODIFICA EM TOKENS OS DADOS

```
32  vocab_size = 10000
33  embedding_dim = 16
34  max_length = 120
35  trunc_type='post'
36  oov_tok = "<OOV>"
37
38  from tensorflow.keras.preprocessing.text import Tokenizer
39  from tensorflow.keras.preprocessing.sequence import pad_sequences
40
41  tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
42  tokenizer.fit_on_texts(training_sentences)
43  word_index = tokenizer.word_index
44  sequences = tokenizer.texts_to_sequences(training_sentences)
45  padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)
46
47  testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
48  testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
49
50  reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

4) TREINA A REDE NEURAL

```python
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()

num_epochs = 10
model.fit(padded, training_labels_final, epochs=num_epochs, validation_data=(testing_padded, testing_labels_final))

e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim) (10000, 16)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 11526
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 171,533
Trainable params: 171,533
Non-trainable params: 0
```

```
Epoch 1/10
782/782 [==============================] - 3s 3ms/step - loss: 0.5014 - accuracy: 0.7361 - val_loss: 0.3439 - val_accuracy: 0.8481
Epoch 2/10
782/782 [==============================] - 3s 3ms/step - loss: 0.2361 - accuracy: 0.9085 - val_loss: 0.3676 - val_accuracy: 0.8414
Epoch 3/10
782/782 [==============================] - 3s 3ms/step - loss: 0.0841 - accuracy: 0.9793 - val_loss: 0.4772 - val_accuracy: 0.8251
Epoch 4/10
782/782 [==============================] - 2s 3ms/step - loss: 0.0194 - accuracy: 0.9978 - val_loss: 0.5367 - val_accuracy: 0.8294
Epoch 5/10
782/782 [==============================] - 3s 3ms/step - loss: 0.0048 - accuracy: 0.9997 - val_loss: 0.6084 - val_accuracy: 0.8276
Epoch 6/10
782/782 [==============================] - 3s 3ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.6463 - val_accuracy: 0.8305
Epoch 7/10
782/782 [==============================] - 2s 3ms/step - loss: 8.3181e-04 - accuracy: 1.0000 - val_loss: 0.6917 - val_accuracy: 0.8308
Epoch 8/10
782/782 [==============================] - 3s 3ms/step - loss: 4.5176e-04 - accuracy: 1.0000 - val_loss: 0.7313 - val_accuracy: 0.8310
Epoch 9/10
782/782 [==============================] - 3s 3ms/step - loss: 2.5692e-04 - accuracy: 1.0000 - val_loss: 0.7702 - val_accuracy: 0.8307
Epoch 10/10
782/782 [==============================] - 3s 3ms/step - loss: 1.5419e-04 - accuracy: 1.0000 - val_loss: 0.8071 - val_accuracy: 0.8307
```

5) SALVA OS VETORES E METADADOS

```
74  import io
75
76  out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
77  out_m = io.open('meta.tsv', 'w', encoding='utf-8')
78  for word_num in range(1, vocab_size):
79    word = reverse_word_index[word_num]
80    embeddings = weights[word_num]
81    out_m.write(word + "\n")
82    out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
83  out_v.close()
84  out_m.close()
```

vecs

-0.014340208	0.009031665	0.031591155	0.0173462	-0.01122054	-0.051589854	-0.053706452	-0.023417432	0.00226255	0.11966835
-0.059547428	-0.017255515	0.03851408	0.029141394	0.0142244855	-0.003947841	-0.006356806	0.0012990927	-0.024790432	0.019497076
-0.04287224	0.0033182232	0.068976924	0.038810205	-0.054081164	0.016296932	-0.042353965	0.040621955	-0.050244257	0.04485376
0.033047333	0.029300667	0.03502349	0.06906605	0.06937742	0.014133729	-0.050950024	0.01630433	-0.010325233	0.030560127
0.07588592	0.010917315	0.037992004	0.002295777	-0.024900965	-0.060219884	0.0038639428	0.04360031	-0.07225229	0.09818568
0.007868569	-0.10522651	-0.026477832	-0.032826614	0.005232549	-0.011066855	0.028658796	-0.0036224544	0.008110191	0.038581442
-0.056746844	0.03097467	-0.012508153	0.032428224	-0.03355766	-0.06555749	-0.0009131555	0.022167284	0.014903704	0.0763061
0.022872664	-0.026746562	0.024507755	0.026136544	0.024124846	0.025745204	0.033787206	0.013035937	-0.058486816	0.08868382
-0.043753274	0.010218633	-0.027006317	0.031058243	-0.040594496	0.051428977	-0.06599921	0.006059604	-0.018853651	0.040145196
0.010191806	0.037186	-0.0012612346	0.013286637	-0.036666945	0.04943981	-0.12168852	0.031361833	-0.0017708178	0.07444053
-0.048742983	0.02339044	0.06589863	-0.019655952	0.011343644	0.010406336	-0.049727295	0.06428928	0.03169412	-0.005446586
-0.053180773	0.029558374	-0.02602737	0.070262656	0.067763954	-0.012867836	-0.01158285	-0.003274816	0.05961246	0.037978828
0.04823384	0.040141888	0.0045990115	-0.0023349298	0.0423587	0.035241824	0.018914111	0.035463516	0.010315613	0.045101985
-0.09842255	-0.0029895403	0.03193144	-0.021507584	0.06721333	0.0010850512	0.04312561	-0.042272013	-0.059112687	-0.021186886
0.0058951955	-0.00389934	0.0670581	0.08345957	0.010070655	0.07795782	0.0014484425	-0.031423565	0.028428232	0.049381353

vecs.tsv

meta

<OOV>
the
and
a
of
to
is
br
in
it
i
this
that
was
as
for
with

meta.tsv

6) TESTE DE SENTENÇA

```
95  sentence = "I really think this is amazing. honest."
96  sequence = tokenizer.texts_to_sequences([sentence])
97  print(sequence)
```

```
[[11, 64, 102, 12, 7, 478, 1200]]
```

B) BASE DE DADOS DE SARCASMO

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

```
import json
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 32
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000
```

1) CARREGANDO OS DADOS DO ARQUIVO SARCASM.JSON

```
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'])
```

2) SEPARANDO OS DADOS EM TREINO E EM TESTE

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

3) CODIFICANDO O TEXTO (TOKENS)

```
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_type, truncating=trunc_type)

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

4) CRIANDO O MODELO DA REDE NEURAL

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

ARQUITETURA DA REDE NEURAL

```
model.summary()
```

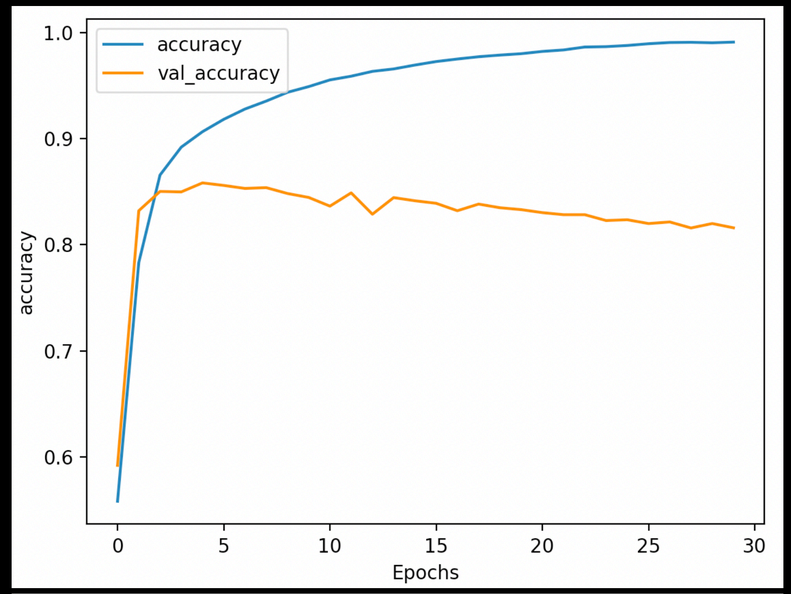
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 32, 16)            160000
_________________________________________________________________
global_average_pooling1d_2 ( (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 24)                408
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 25
=================================================================
Total params: 160,433
Trainable params: 160,433
Non-trainable params: 0
```

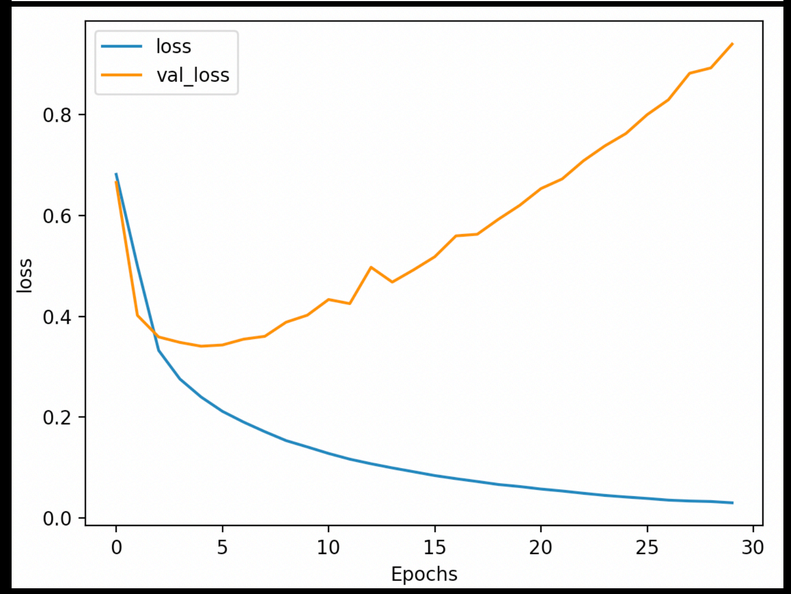
5) TREINANDO A REDE NEURAL

```
num_epochs = 30

history = model.fit(training_padded, training_labels, epochs=num_epochs,
                    validation_data=(testing_padded, testing_labels), verbose=2)
```

6) PLOTANDO O DESEMPENHO DA REDE NEURAL

```python
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, "acc")
plot_graphs(history, "loss")
```

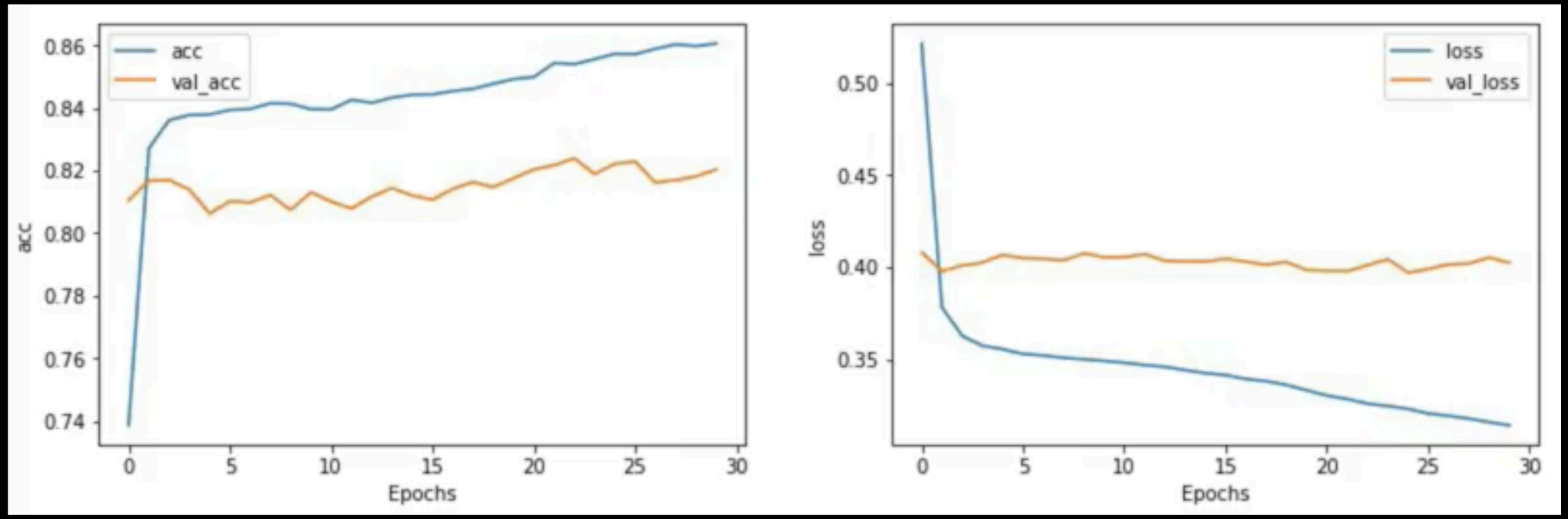




Como podemos melhorar
a acurácia da validação?

ALTERANDO VALORES CONFIGURADOS...

```
vocab_size = 1000    (was 10,000)
embedding_dim = 16
max_length = 16      (was 32)
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000
```



LINKS ÚTEIS

http://ai.stanford.edu/~amaas/data/sentiment/

http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf

https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%202%20-%20Lesson%201.ipynb

https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%202%20-%20Lesson%202.ipynb

https://github.com/tensorflow/datasets/blob/master/docs/catalog/imdb_reviews.md

https://www.tensorflow.org/datasets/api_docs/python/tfds/features/text/SubwordTextEncoder

EXERCÍCIO

Iremos reutilizar a base de dados de notícias da BBC criado na última aula. Nesta base de dados, artigos são classificados em categorias.

Tente implementar uma rede neural que possa ser treinada com esta base de dados para determinar quais palavras determinam sua categoria.

Crie os arquivos vecs.tsv e meta.tsv, conforme exemplos anteriores e os carreguem no projetor de embedding.

https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%202%20-%20Exercise%20-%20Question.ipynb

DÚVIDAS?

RAFAELVC2@GMAIL.COM