

## Introdução à NLP e RNN:

### 1. Introdução

Nesta aula, vamos estudar os aspectos básicos de NLP (Natural Language Processing) e RNN (Recurrent Neural Networks). NLP é uma área dentro da Inteligência Artificial que se dedica a desenvolver a capacidade da máquina de entender a linguagem humana.

Até agora, trabalhamos somente com aplicações de aprendizado de máquina em imagens. Com isso, possivelmente não fazemos a mínima ideia de como redes neurais conseguem processar frases. Esta apostila vai dar uma ideia de como a linguagem natural pode ser processada por redes neurais.

Vamos seguir o curso introdutório: “NLP Zero to Hero playlist” <https://goo.gle/nlp-z2h>.

Na descrição de cada vídeo desse curso, há um link para Notebook Colab correspondente.

### 2. Tokenização

O primeiro passo de um sistema NLP é a tokenização. Tokenização converte as palavras de uma frase numa sequência de números, associando uma token (número) a cada palavra. Tokenizando a frase: “I love my dog”, obtemos o código: “1 2 3 4”, atribuindo um token para cada palavra:

Tabela 1: Tokenização associa um número (token) para cada palavra.

Palavra	I	love	my	dog
Código	1	2	3	4

Agora, se fosse codificar a frase: “I love my cat”, teríamos o código: “1 2 3 5”. As três primeiras palavras já receberam códigos na frase anterior. Apenas a palavra “cat” recebe um novo token, pois não ainda não tem token associado.

Tabela 2: Tokenização reutiliza os tokens gerados para as frases anteriores.

Palavra	I	love	my	cat
Código	1	2	3	5

Notebook original do curso: <https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%201.ipynb#scrollTo=zX4Kg8DUTKWO>

Em Keras, a tokenização explicada acima pode ser codificado como:

```
#part1.py - Licensed under the Apache License, Version 2.0
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
    'i love my dog',
    'I love my cat',
    'You love my dog!'
]

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
print(tokenizer.word_index)
```

Programa 1: Tokenização em Keras gera um dicionário que associa cada palavra a um número.

<https://colab.research.google.com/drive/1SdmCMXdZAq4A1xS5uY4s9MFPuUCoGL2t?usp=sharing> ~/deep/algpi/nlp/part1.py

O método *fit\_on\_texts* efetua a tokenização. A variável *tokenizer* será armazenada como um dicionário de Python, associando cada palavra a um número.

{'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6}

O programa 1 vai tokenizar as 100 palavras mais frequentes nas frases de *sentences* (*num\_words* = 100). Tokenizer ignora maiúscula/minúscula e os sinais de pontuação (como o ponto de exclamação de "dog!"). Tokenização também ignora "apóstrofo-s" (ex: "granny's" é igual a "granny").

*Curiosidade:* Se perguntar a ChatGPT numa seção nova (07/10/2024):

"How many f's are there in the word fluffy?"

Ela responde:

"The word "fluffy" contains **two** "f"s."

ChatGPT e outros modelos de linguagem usualmente trabalham com tokens para textos mais complexos (podem trabalhar letra a letra para problemas simples). Assim, podem errar se fizer perguntas sobre as letras de uma palavra. Fiz a mesma pergunta para Gemini e Llama e ambos acertaram.

### 3. Sequenciamento

O sequenciamento converte texto em sequência de tokens.

Notebook original do curso: <https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%202.ipynb#scrollTo=ArOPfBwyZth>

```
1 #part2.py - Licensed under the Apache License, Version 2.0
2 import tensorflow as tf
3 from tensorflow import keras
4
5 from tensorflow.keras.preprocessing.text import Tokenizer
6 from tensorflow.keras.preprocessing.sequence import pad_sequences
7
8 sentences = [
9     'I love my dog',
10    'I love my cat',
11    'You love my dog!',
12    'Do you think my dog is amazing?'
13 ]
14
15 tokenizer = Tokenizer(num_words = 100, oov_token=<OOV>)
16 tokenizer.fit_on_texts(sentences)
17 word_index = tokenizer.word_index
18
19 sequences = tokenizer.texts_to_sequences(sentences)
20
21 padded = pad_sequences(sequences, maxlen=5)
22 print("\nWord Index = ", word_index)
23 print("\nSequences = ", sequences)
24 print("\nPadded Sequences:")
25 print(padded)
26
27 # Try with words that the tokenizer wasn't fit to
28 test_data = [
29     'i really love my dog',
30     'my dog loves my manatee'
31 ]
32
33 test_seq = tokenizer.texts_to_sequences(test_data)
34 print("\nTest Sequence = ", test_seq)
35
36 padded = pad_sequences(test_seq, maxlen=10)
37 print("\nPadded Test Sequence: ")
38 print(padded)
```

Programa 2: Programa em Keras que converte frases em sequências de tokens.

[https://colab.research.google.com/drive/17AxfyA6S566-xrhj3FWRTTfnUxgRQ\\_e4?usp=sharing](https://colab.research.google.com/drive/17AxfyA6S566-xrhj3FWRTTfnUxgRQ_e4?usp=sharing) ~deep/algpi/nlp/part2.py

O método *fit\_on\_texts* (linha 16) efetua a tokenização, atribuindo tokens às 100 palavras mais frequentes nas frases de *sentences*, e acrescenta o token <OOV> “out of vocabulary” que indica as palavras que não constam no vocabulário:

```
Word Index = {'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}
```

O método *texts\_to\_sequences* (linha 19) converte as 4 frases de *sentences* em sequências de tokens *sequences*:

```
Sequences = [[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
```

Por exemplo, a sequência [5, 3, 2, 4] significa “I love my dog”.

Uma rede neural possui um número fixo de entradas. Como fazer com que as sentenças tenham o mesmo comprimento para alimentar uma rede neural? A solução mais simples é fazer “padding”. A função `pad_sequences` (linha 21) preenche as frases com “0” no começo para que todas as sequências tenham comprimento  `maxlen=5`.

Nota: Se o parâmetro  `maxlen` não for especificado,  `maxlen` torna-se o comprimento da frase mais longa.

Nota: Parece que as técnicas mais avançadas utilizam outras técnicas como “ragged tensor” que pode ter um número variável de elementos em alguma dimensão.

Padded Sequences:

```
[[ 0  5  3  2  4]
 [ 0  5  3  2  7]
 [ 0  6  3  2  4]
 [ 9  2  4 10 11]]
```

Nota: Se quiser preencher as frases com “0” no fim das sequências (em vez de no início), deve passar o argumento `padding='post'`.

Nota: A última frase que tinha inicialmente 7 palavras foi truncada (duas palavras iniciais foram apagadas) para resultar numa frase com 5 palavras. É possível para apagar as palavras finais com `truncating='post'`.

Codificando as novas sentenças (linhas 27-34):

```
'i really love my dog',
'my dog loves my manatee'
```

obtemos (linha 34):

```
Test Sequence = [[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]
```

onde o token “1” indica `<OOV>` ou palavras que não constam no vocabulário. As palavras “really”, “loves” e “manatee” foram codificadas como `<OOV>`.

Nota: Manatee é peixe-boi.

Fazendo `padding='pre'` nessas duas frases para que tenham comprimento 10 (linha 36), obtemos:

Padded Test Sequence:

```
[[0 0 0 0 5 1 3 2 4]
 [0 0 0 0 2 4 1 2 1]]
```

## 4. Embedding

Seguindo o tutorial de TensorFlow, vamos usar o conjunto de dados “Sarcasm in News Headlines” por Rishabh Misra para classificar títulos de notícias em sarcástico ou não.

[rishabhmisra.github.io/publications](http://rishabhmisra.github.io/publications)

<https://storage.googleapis.com/learning-datasets/sarcasm.json>

Nota: Esse conjunto de dados é pequeno demais (5 MB, com somente 26k frases) e o problema é simples demais, de forma que os experimentos não ilustram bem os conceitos. Há várias palavras que aparecem uma única vez nesse conjunto (por exemplo, “granny”), o que torna impossível criar um bom modelo. Precisaria achar outro problema mais adequado com um conjunto de treino maior.

Esse conjunto está armazenado no arquivo “sarcasm.json” com 3 campos:

“article\_link”: link para o artigo (não vamos usar),

“headline”: o título do artigo,

“is\_sarcastic”: 1 se for sarcástico e 0 caso contrário.

Os dois primeiros registros desse conjunto são:

```
[  
{"article_link": "https://www.huffingtonpost.com/entry/versace-black-  
code_us_5861fbef4b0de3a08f600d5", "headline": "former versace store clerk sues over secret  
'black code' for minority shoppers", "is_sarcastic": 0},  
 {"article_link": "https://www.huffingtonpost.com/entry/roseanne-revival-  
review_us_5ab3a497e4b054d118e04365", "headline": "the 'roseanne' revival catches up to our  
thorny political mood, for better and worse", "is_sarcastic": 0},  
 (...)
```

Figura 1: Formato do conjunto de textos “Sarcasm in News Headlines”.

<https://storage.googleapis.com/learning-datasets/sarcasm.json>

O objetivo é classificar os títulos dos artigos em “sarcástico” ou “não-sarcástico”.

Vamos normalizar todas as frases para terem 100 palavras. Frases curtas serão preenchidas com “0” no final para terem 100 palavras. As frases que contêm mais de 100 palavras serão truncadas no final.

Vamos construir um dicionário com as 10.000 palavras mais usadas no conjunto de treino. As palavras que não constam nesse dicionário serão codificadas como “1” ou <OOV>.

Vamos usar a técnica “embedding”, isto é, vamos associar um vetor de 16 dimensões para cada uma das 10.000 palavras (tokens). Os  $10.000 \times 16$  pesos serão calculadas pela retro-propagação de forma a facilitar classificar as frases em “sarcástica” ou não. Isto consiste em associar, para cada palavra, 16 atributos que ajudem a classificar a frase. Esta associação é feita usando uma camada de embedding com uma matriz de  $10.000 \times 16$  parâmetros que indica os atributos para cada uma das 10.000 palavras. Essa matriz é inicializada aleatoriamente e treinada fazendo retro-propagação. Embedding extrai atributos que ajudam a classificar título como sarcástico ou não.

Podemos usar a seguinte intuição para entender “embedding”. As palavras que aparecem frequentemente em títulos sarcásticos recebem um valor alto nos atributos que indicam título sarcástico. Um título com muitas palavras com altos valores de nos atributos sarcásticos, será classificado como sarcástico.

Por outro lado, pode ser que uma palavra P seja típica de título sarcástico mas a presença de uma outra palavra Q na mesma frase torne a frase não-sarcástica. Para poder contemplar casos como este, vários atributos de embedding (no nosso caso 16) são extraídos de cada palavra.

Durante a predição, dada uma frase com  $n$  palavras, vamos convertê-la numa sequência de 100 tokens (fazendo padding ou cortando palavras da frase para ter exatamente 100 tokens). Depois, vamos calcular os 16 atributos de embedding para cada uma das 100 palavras. Em seguida, vamos calcular a média vetorial dos atributos das 100 palavras (global average pooling), obtendo um vetor com 16 elementos. Esses 16 números vão entrar numa rede densa com uma única saída que classificará a frase em “sarcástica” ou não.

Nota: Como estamos usando global average pooling, a ordem das palavras na frase não afeta a classificação.

A figura 2 mostra graficamente como uma frase com palavras é transformada até chegar à saída final que permite classificar a frase.

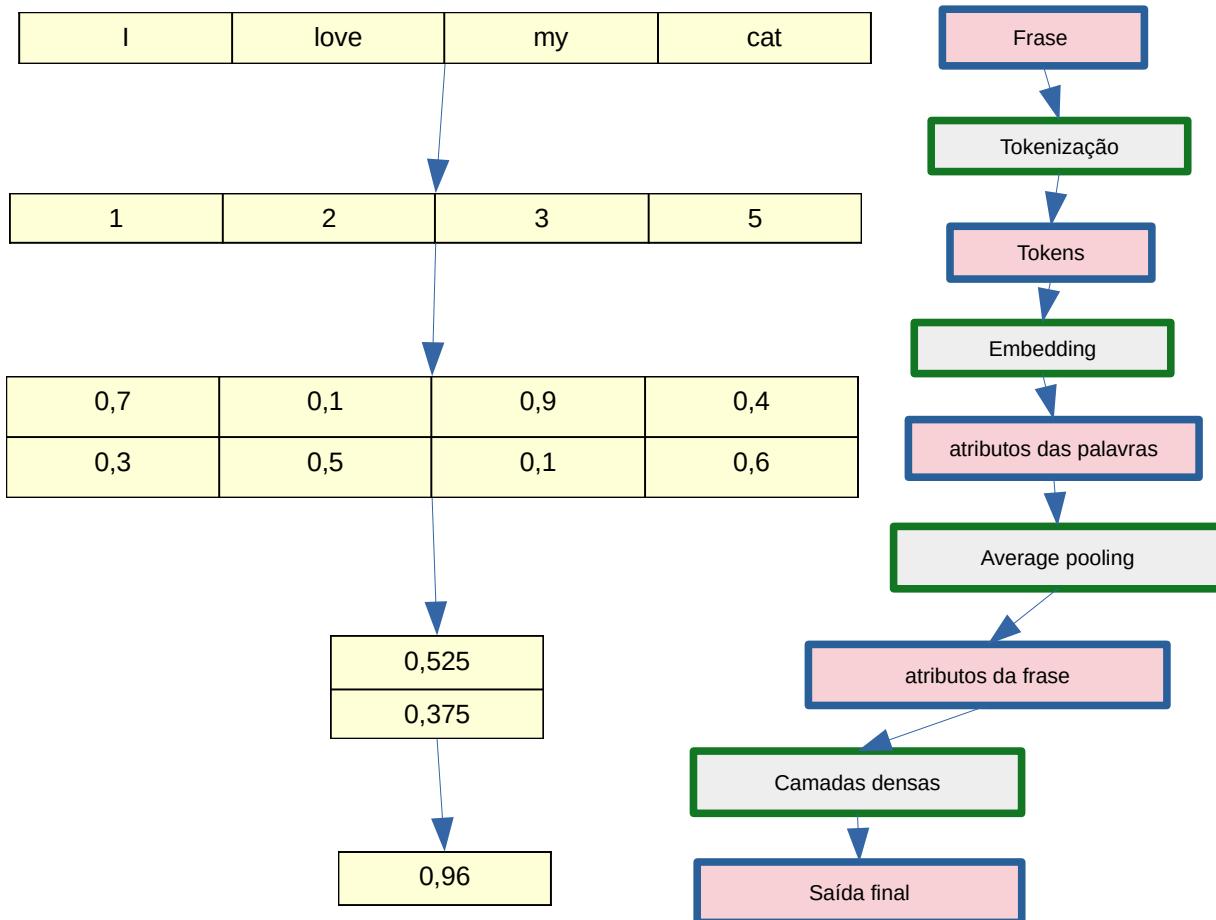


Figura 2: Como uma frase é classificada no programa 3.

O programa abaixo implementa as técnicas discutidas. A acurácia de treino é 99.30% mas a acurácia de teste é 80.92% (overfitting).

```
1 #!pip uninstall tensorflow -y --quiet
2 #!pip install tensorflow==2.15 --quiet
3 !wget --no-check-certificate \
4     https://storage.googleapis.com/learning-datasets/sarcasm.json \
5     -O ./sarcasm.json
```

```
1 #part3train.py - Licensed under the Apache License, Version 2.0
2 import os, json, sys
3 import tensorflow as tf
4 from tensorflow.keras.preprocessing.text import Tokenizer
5 from tensorflow.keras.preprocessing.sequence import pad_sequences
6 import pickle
7
8 vocab_size = 10000; embedding_dim = 16; max_length = 100
9 trunc_type='post'; padding_type='post'; oov_tok = "<OOV>"
10 training_size = 20000
11
12 with open("sarcasm.json", 'r') as f:
13     datastore = json.load(f)
14
15 sentences = []
16 labels = []
17 for item in datastore:
18     sentences.append(item['headline'])
19     labels.append(item['is_sarcastic'])
20
21 training_sentences = sentences[0:training_size]
22 testing_sentences = sentences[training_size:]
23 training_labels = labels[0:training_size]
24 testing_labels = labels[training_size:]
25 print("Training size=",len(training_labels),"Testing size=",len(testing_labels))
26
27 tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
28 tokenizer.fit_on_texts(training_sentences)
29
30 word_index = tokenizer.word_index
31 import more_itertools; print(more_itertools.take(10, word_index.items()))
32
33 print(training_sentences[0])
34 training_sequences = tokenizer.texts_to_sequences(training_sentences)
35 print(training_sequences[0])
36 training_padded = pad_sequences(training_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
37 print(training_padded[0])
38
39 testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
40 testing_padded = pad_sequences(testing_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
41
42 import numpy as np
43 training_padded = np.array(training_padded); training_labels = np.array(training_labels)
44 testing_padded = np.array(testing_padded); testing_labels = np.array(testing_labels)
45
46 model = tf.keras.Sequential([
47     tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
48     tf.keras.layers.GlobalAveragePooling1D(),
49     tf.keras.layers.Dense(24, activation='relu'),
50     tf.keras.layers.Dense(1, activation='sigmoid')
51 ])
52 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']); model.build()
53 from tensorflow.keras.utils import plot_model
54 plot_model(model, to_file='part3.png', show_shapes=True);
55 model.summary()
56
57 num_epochs = 30
58 history = model.fit(training_padded, training_labels, epochs=num_epochs, validation_data=(testing_padded, testing_labels),
59 verbose=2)
60
61 import matplotlib.pyplot as plt
62 def plot_graphs(history, string):
63     plt.plot(history.history[string]); plt.plot(history.history['val_'+string])
64     plt.xlabel("Epochs"); plt.ylabel(string)
65     plt.legend([string, 'val_'+string])
66     plt.show()
67 plot_graphs(history, "accuracy"); plot_graphs(history, "loss")
68
69 with open('sarcasm.pkl', 'wb') as handle:
70     pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
71 model.save("sarcasm.keras")
```

Programa 3: Treina rede sarcasm.keras para classificar os títulos de notícias em sarcásticos ou não.

<https://colab.research.google.com/drive/136zIEsQf84em9AhjZy8y8BeCQSSG3ie?usp=sharing> ~deep/algpi/nlp/part3train.py

O programa 3 acima (part3train.py) treina a rede neural. Rodando-o, obtemos na saída:

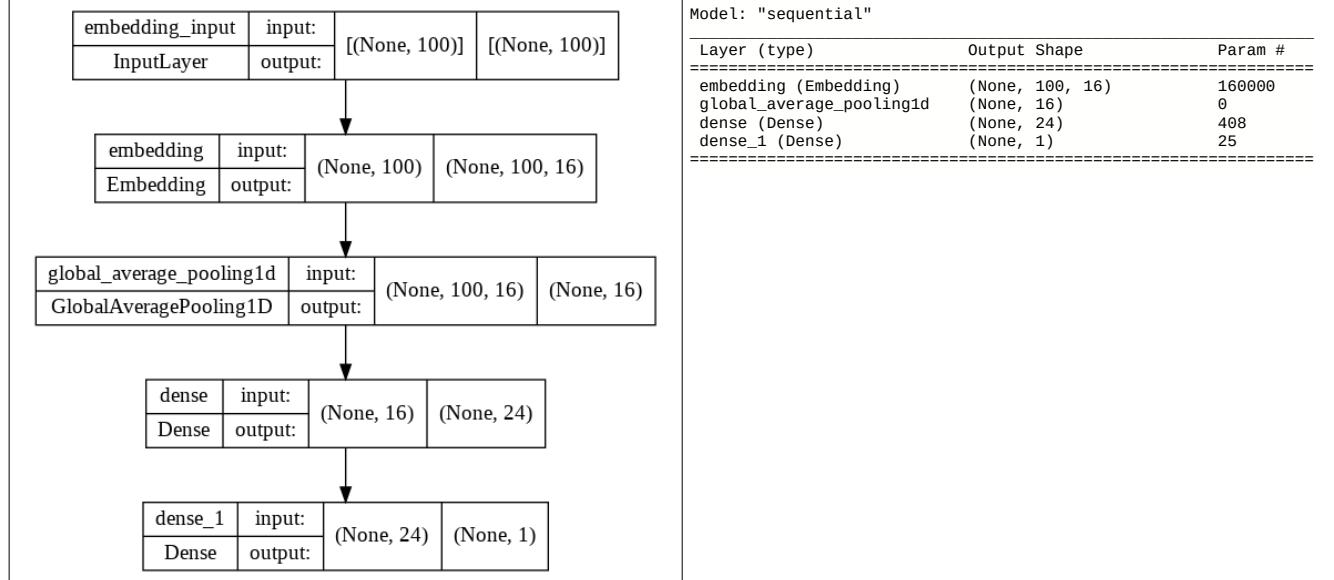


Figura 3: Estrutura da rede do programa 3.

```
[Número de frases] Training size= 20000 Testing size= 6709
[Os 10 primeiros tokens (os 10 mais usados)] ['<OOV>', 1), ('to', 2), ('of', 3), ('the', 4), ('in', 5), ('for', 6),
('a', 7), ('on', 8), ('and', 9), ('with', 10)]
[A frase para classificar] "former versace store clerk sues over secret 'black code' for minority shoppers"
[A frase como sequência de tokens] [328, 1, 799, 3405, 2404, 47, 389, 2214, 1, 6, 2614, 8863]
[Como sequência de tokens com padding] [ 328 1 799 3405 2404 47 389 2214 1 6 2614 8863 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 ]
```

Epoch 1/30 - 2s - loss: 0.66664 - accuracy: 0.5872 - val\_loss: 0.6028 - val\_accuracy: 0.7468 - 2s/epoch - 4ms/step  
Epoch 10/30 - 1s - loss: 0.1246 - accuracy: 0.9562 - val\_loss: 0.4291 - val\_accuracy: 0.8416 - 1s/epoch - 2ms/step  
Epoch 20/30 - 1s - loss: 0.0529 - accuracy: 0.9837 - val\_loss: 0.7421 - val\_accuracy: 0.8193 - 1s/epoch - 2ms/step  
Epoch 30/30 - 1s - loss: 0.0242 - accuracy: 0.9930 - val\_loss: 1.0867 - val\_accuracy: 0.8092 - 1s/epoch - 2ms/step

Figura 4: Saída do programa 3.

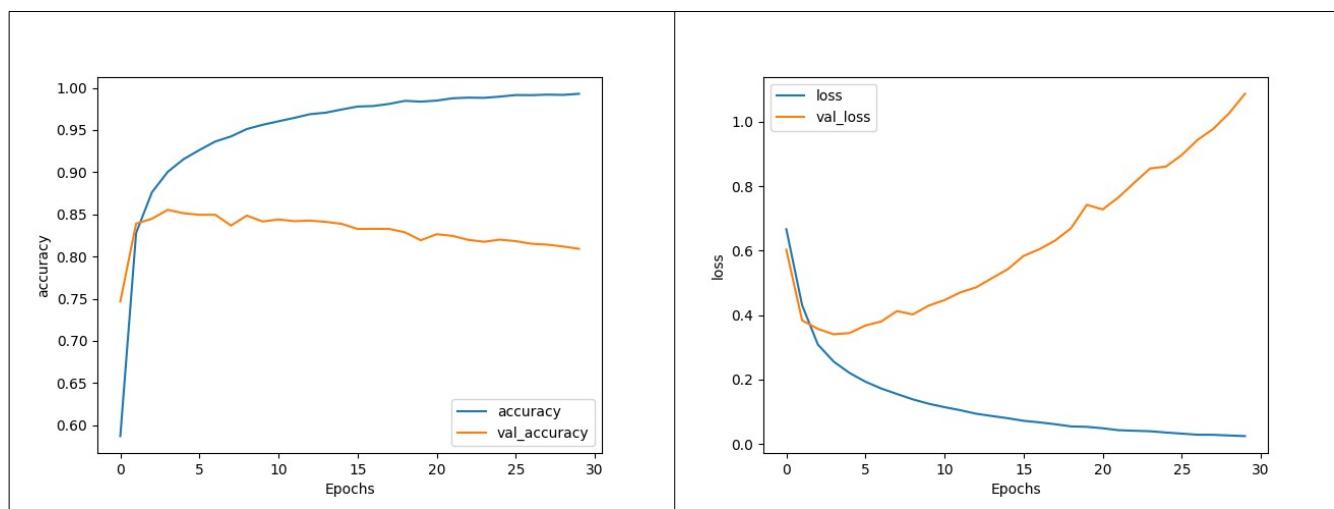


Figura 5: Acuracidade e perda do programa 3 ao longo das épocas.

No programa 3, a primeira impressão (linha 25) mostra o número de frases de treino e de teste:

Training size= 20000 Testing size= 6709.

A segunda impressão (linha 31) mostra as 10 primeiras palavras do dicionário (com total de 10.000 palavras):

[('<OOVS', 1), ('to', 2), ('of', 3), ('the', 4), ('in', 5), ('for', 6), ('a', 7), ('on', 8), ('and', 9), ('with', 10)]

As impressões seguintes (linhas 33, 35 e 37) mostram como a frase:

"former versace store clerk sues over secret 'black code' for minority shoppers"

será codificada como sequência de tokens:

[328, 1, 799, 3405, 2404, 47, 389, 2214, 1, 6, 2614, 8863]

e preenchida com padding “0” no fim para ficar com comprimento 100.

```
[ 328   1  799 3405 2404   47   389 2214   1   6 2614 8863   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 ]
```

A primeira camada da rede (linha 47) faz “embedding”, isto é, recebe uma frase com 100 tokens e transforma cada token em 16 atributos. Ou seja, dado um token  $P$  (um número inteiro que representa uma palavra) e uma tabela de embedding  $T$  com  $10.000 \times 16 = 160.000$  pesos, os 16 atributos de embedding de  $P$  estão na linha  $P$  da matriz  $T$ , isto é,  $T[P]$ . Repetindo este processo para cada palavra de uma frase com 100 palavras, a frase será convertida numa matriz com  $100 \times 16$  atributos.

A segunda camada “global\_average\_pooling” (linha 48), tira a média aritmética dos atributos dos 100 tokens, transformando os  $100 \times 16$  atributos num único vetor com 16 elementos.

Depois, seguem duas camadas densas, que transformam 16 atributos numa única saída com “probabilidade” do título ser sarcástico.

O programa 4 (part3test.py) abaixo faz predição usando o modelo *sarcasm.keras* construído anteriormente.

*Tokenizer* não faz parte do modelo da rede e é necessário salvá-lo num arquivo separado em formato pickle usado para serializar objetos de Python: *sarcasm.pkl*.

```

1 #part3test.py - Licensed under the Apache License, Version 2.0
2 import os, json, sys
3 import tensorflow as tf
4 from tensorflow.keras.preprocessing.text import Tokenizer
5 from tensorflow.keras.preprocessing.sequence import pad_sequences
6 from tensorflow.keras.models import load_model
7 import pickle
8
9 vocab_size = 10000
10 embedding_dim = 16
11 max_length = 100
12 trunc_type='post'
13 padding_type='post'
14 oov_tok = "<OOV>"
15 training_size = 20000
16
17 model=load_model("sarcasm.keras")
18 with open('sarcasm.pkl', 'rb') as handle:
19     tokenizer = pickle.load(handle)
20
21 e = model.layers[0]
22 weights = e.get_weights()[0]
23 print(weights.shape) # shape: (vocab_size, embedding_dim)
24
25 qx = [
26     "granny starting to fear spiders in the garden might be real",
27     "game of thrones season finale showing this sunday night",
28     "russian strike on rail station kills 25 - Ukraine",
29     "biden to forgive $10k in student loans - in unrelated news, nation's colleges raise tuition by $10k",
30     "Internet cut off as protesters shared images of police brutality. CNN investigates",
31     "China Starting To Worry TikTok Has Made Americans Even Dumber Than They Intended",
32     "Dad Again Uses Kids As Excuse To Get McDonald's",
33     "Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days",
34     "Chinese city 'stretched to the limit' as millions wait in line for Covid tests in extreme heat",
35     "Why Tesla's stock is so much cheaper today",
36     "Spectacular images revealed in weather photography competition",
37     "weather competition images spectacular in revealed photography",
38     "Cruise ship rescues Cuban migrants stranded at sea"
39 ]
40 gy = [1,0,0,1,0,1,1,0,0,0,0,0]
41 sequences = tokenizer.texts_to_sequences(qx)
42 padded = pad_sequences(sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
43 qp=model.predict(padded)
44 for i,p,y,s in zip(range(len(qx)),qp,gy,qx):
45     print("%2d %4.2f %1d %s"%(i,p[0],y,s))

```

Programa 4: Usa a rede *sarcasm.keras* gerada pelo programa 3 para classificar títulos de notícias em sarcásticos ou não.

<https://colab.research.google.com/drive/136zIEsQf84em9AhjZy8y8BeCQSsG3gie?usp=sharing> ~deep/algpi/nlp/part3test.py

Rodando esse programa em 12 frases obtemos:

```

Pesos da primeira camada: (10000, 16)
QP QY QX
0 0.99 1 granny starting to fear spiders in the garden might be real
1 0.00 0 game of thrones season finale showing this sunday night
2 0.00 0 russian strike on rail station kills 25 - Ukraine
3 1.00 1 biden to forgive $10k in student loans - in unrelated news, nation's colleges raise tuition by $10k
4 0.01 0 Internet cut off as protesters shared images of police brutality. CNN investigates
5 1.00 1 China Starting To Worry TikTok Has Made Americans Even Dumber Than They Intended
6 0.00 1 Dad Again Uses Kids As Excuse To Get McDonald's
7 0.00 1 Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days
8 1.00 0 Chinese city 'stretched to the limit' as millions wait in line for Covid tests in extreme heat
9 1.00 0 Why Tesla's stock is so much cheaper today
10 0.52 0 Spectacular images revealed in weather photography competition
11 0.52 0 weather competition images spectacular in revealed photography
12 0.00 0 Cruise ship rescues Cuban migrants stranded at sea

```

A primeira impressão (linha 23) mostra o formato dos pesos da camada de embedding ( $10000 \times 16$ ). Estes  $10000 \times 16$  pesos constituem a matriz de *embedding* e associa um vetor de 16 elementos para cada uma das 10000 palavras do vocabulário.

As primeiras 3 colunas das linhas impressas seguintes (linhas 44-45) mostram o índice da frase de teste, a predição da rede  $QP$ , a saída ideal  $QY$ , e a frase de entrada  $QX$  para cada uma das 12 frases de teste.

As frases 10 e 11 contêm as mesmas palavras em ordens diferentes. A rede atribuiu exatamente a mesma nota para ambas frases, pois o modelo construído não leva em consideração a ordem das palavras. Este modelo analisa cada palavra individualmente como sendo típica de notícia sarcástica ou não, não importando a ordem em que elas aparecem na frase.

*Nota:* Tenho a impressão de que tanto o espaço de embedding de dimensão 16, como a camada densa escondida com 24 neurônios do programa 3 são exageradamente altos para este problema. Testei executar o mesmo programa com dimensão de embedding 2 e camada escondida com 2 neurônios e obtive acuracidade de treino de 0.9067 acuracidade de teste 0.8562.

[https://colab.research.google.com/drive/1aSkATunvvoU7HKd5EHrQDDoauZGXCYd#scrollTo=xn55\\_SdpLPO2](https://colab.research.google.com/drive/1aSkATunvvoU7HKd5EHrQDDoauZGXCYd#scrollTo=xn55_SdpLPO2)  
Tínhamos obtido, no programa 3, acuracidade de treino 0.9930 e acuracidade de teste 0.8092. Isto é, mudando os parâmetros, overfitting diminuiu e acuracidade de teste aumentou.

*Exercício:* Procure na internet 5 títulos de notícias “normais” e 5 títulos sarcásticos em inglês e verifique se a rede consegue categorizá-los corretamente.

*Exercício:* Imprima o vetor de atributos de embedding de algumas palavras típicas de manchetes sarcásticas e outras não-sarcásticas. Você consegue identificar os atributos que caracterizam palavras de manchetes sarcásticas? Isto fica mais fácil se a dimensão de embedding for um.

Solução: [A completar.]

A solução privada está em `~/deep/algpi/nlp/part36train.py` e `part36test.py`.

A conclusão final é que não dá para caracterizar palavras típicas de títulos sarcásticos e não-sarcásticos pois o conjunto de treino é pequeno demais.

A frase:

`granny starting to fear spiders in the garden might be real`

Foi associada aos atributos de embedding:

`0.14 1.57 0.42 0.01 0.14 -0.07 -2.28 2.07 -0.88 0.22 -0.25`

onde atributo acima de -0.03 (aproximadamente) é sarcástico.

A palavra “starting” foi classificada como uma palavra altamente sarcástica, pois apareceu em vários títulos sarcásticos do conjunto de treino.

A palavra “spider” foi classificada como um pouco sarcásticas. Ela aparece uma única vez num título não sarcástico. Não sei explicar por que “spider” foi classificada como palavra um pouco sarcástica, se ela só apareceu em títulos não-sarcásticos.

## 5. RNN (Recurrent Neural Network)

*Nota:* Tipicamente RNNs são usadas em NLP. Assim, usando-a em títulos sarcásticos a taxa de acerto deveria aumentar. Porém, a taxa de acerto diminui! Isto acontece provavelmente porque o conjunto de treino é pequeno demais. Assim, os exemplos abaixo ficam meio artificiais, mas servem para ilustrar os conceitos.

Usando somente camada *embedding*, uma frase é classificada sem considerar a ordem das palavras na frase. RNN (Recurrent Neural Network) considera a ordem das palavras.

Vamos acompanhar o blog introdutório:

<https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/>  
para descobrir como funciona uma RNN, tipicamente usada em NLP.

A figura abaixo mostra uma RNN (SimpleRNN de Keras):

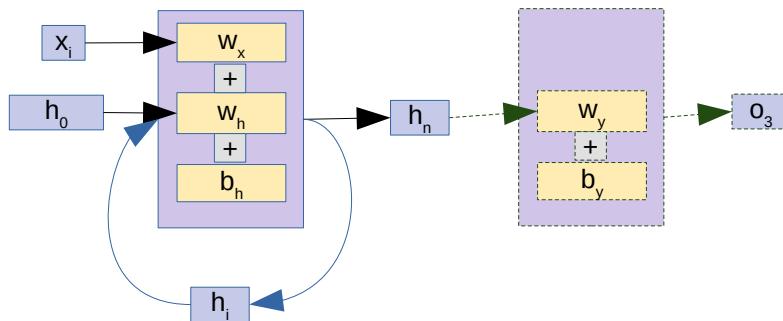


Figura F: Funcionamento de RNN.

A saída de RNN é retro-alimentada na entrada. A rede RNN contém um estado escondido  $h$  (um vetor de números) que altera o seu comportamento. A cada iteração do neurônio, um novo  $x_i$  (uma palavra da frase) entra na rede e a rede gera uma nova saída  $h_i$  que é o seu estado escondido. A saída final da camada RNN é o seu último estado escondido  $h_n$ . Na prática, costuma ter uma ou mais camadas densas após RNN (camada tracejada da figura F).

Sejam  $n$  a quantidade de iterações (o número de palavras na frase), e  $d$  a dimensão das variáveis de entrada (a dimensão do espaço de embedding ou o número de atributos das palavras),  $m$  a dimensão do vetor escondido (também a dimensão do vetor de saída). Então:

Nota: as variáveis em azul são da camada densa final e não fazem propriamente parte da camada RNN.

- Input:  $x_i \in \mathbb{R}^d$ ,  $0 \leq i < n$ . São as  $n$  palavras convertidas em  $n$  vetores de embedding com dimensão  $d$  cada um.
- Hidden unit:  $h_i \in \mathbb{R}^m$ ,  $0 \leq i \leq n$ . O vetor  $h_i$  com  $m$  elementos representa o estado escondido.
- Weights for input units:  $w_x \in \mathbb{R}^{d \times m}$ . Formam a matriz de pesos que irá multiplicar os vetores de embedding.
- Weights for hidden units:  $w_h \in \mathbb{R}^{m \times m}$ . Formam a matriz de pesos que irá multiplicar o vetor escondido anterior.
- Bias for hidden units:  $b_h \in \mathbb{R}^m$ .
- Weights for dense layer:  $w_y \in \mathbb{R}^{m \times 1}$
- Bias for dense layer:  $b_y \in \mathbb{R}^1$

- Final output  $o_3 \in \mathbb{R}$

## Simulação da camada SimpleRNN de Keras

Em Keras, há a camada SimpleRNN que implementa RNN. Para entender exatamente o que acontece dentro dessa camada, vamos tentar emular numericamente o seu funcionamento.

Vamos “desatar” o loop quando RNN executa  $n=3$  iterações:

*Nota:* Penso que provavelmente RNN foi originalmente projetada para ser executada iterativamente dentro de um loop. Porém, por algum requisito de implementação (talvez para poder paralelizar usando GPU), o laço foi “desatado” na implementação atual de SimpleRNN de Keras.

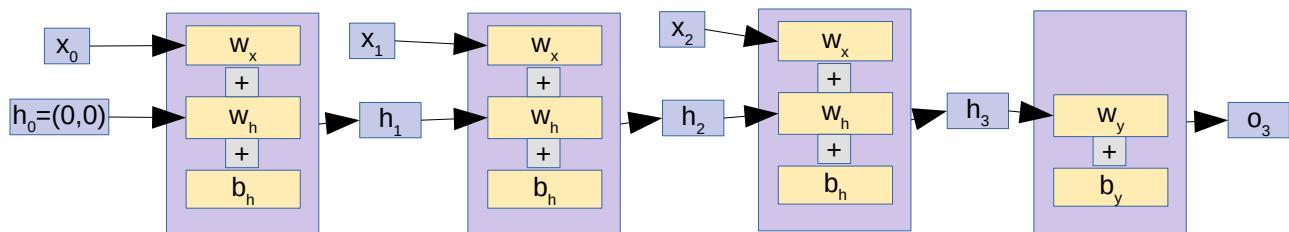


Figura: RNN com laço desatado. As variáveis  $w_x$ ,  $w_h$  e  $b_h$  são iguais nas três primeiras caixas.

O seguinte código Python efetua os cálculos que uma camada RNN executaria para  $n=3$  (iterações ou número de palavras na frase  $[x_0, x_1, x_2]$ ),  $d=2$  (dimensão de embedding de entrada) e  $m=4$  (dimensão de saída e do estado escondido):

```

h0 = np.zeros(4)
h1 = x[0]@wx + h0@wh + bh
h2 = x[1]@wx + h1@wh + bh
h3 = x[2]@wx + h2@wh + bh
o3 = h3@wy + by

```

Onde “ $@$ ” é multiplicação matricial (entre duas matrizes; entre vetor e matriz; ou entre dois vetores).

Vamos verificar se a implementação de SimpleRNN de Keras é exatamente igual às equações acima. O programa abaixo faz duas contas:

- a) Usa uma SimpleRNN para  $n=3$ ,  $d=2$  e  $m=4$ .
- b) Faz as mesmas contas “manualmente” usando as equações acima.

```

#rnn3.py (~/deep/algpi/nlp/rnn3.py)
#https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/
from pandas import read_csv
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import math
import matplotlib.pyplot as plt

m=4 #Hidden variables and output dimension
n=3 #Iterations or number of words
d=2 #Input dimension or embedding dimension
print("m=%d n=%d d=%d"%(m,n,d))

def create_RNN():
    model = Sequential()
    model.add(SimpleRNN(m, input_shape=(n,d), activation="linear"))
    model.add(Dense(units=1, activation="linear"))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

demo_model = create_RNN()

from tensorflow.keras.utils import plot_model
plot_model(demo_model, to_file='rnn3.png', show_shapes=True);
demo_model.summary()

wx = demo_model.get_weights()[0]; print('wx = ', wx)
wh = demo_model.get_weights()[1]; print('wh = ', wh)
bh = demo_model.get_weights()[2]; print('bh = ', bh)
wy = demo_model.get_weights()[3]; print('wy = ', wy)
by = demo_model.get_weights()[4]; print('by = ', by)

# Make prediction using the network
x = np.ones((n,d),dtype=np.float32)
# Reshape the input to the required d X n (embedding dim X number of words)
x_input = np.reshape(x,(1, n, d))
print("x_input=",x_input); print("x[0]=",x[0]); print("x[1]=",x[1]); print("x[2]=",x[2])
y_pred_model = demo_model.predict(x_input,verbose=0)
y_pred_model = np.squeeze(y_pred_model, axis=0)

# Make manually the same computation
h0 = np.zeros(m); print("h0 = ",h0)
h1 = x[0]@wx + h0 + bh; print('h1 = ',h1)
h2 = x[1]@wx + h1@wh + bh; print('h2 = ',h2)
h3 = x[2]@wx + h2@wh + bh; print('h3 = ',h3)
o3 = h3@wy + by; print('o3 = ',o3)

print("Prediction from our manual computation ", o3)
print("Prediction from the network ", y_pred_model)

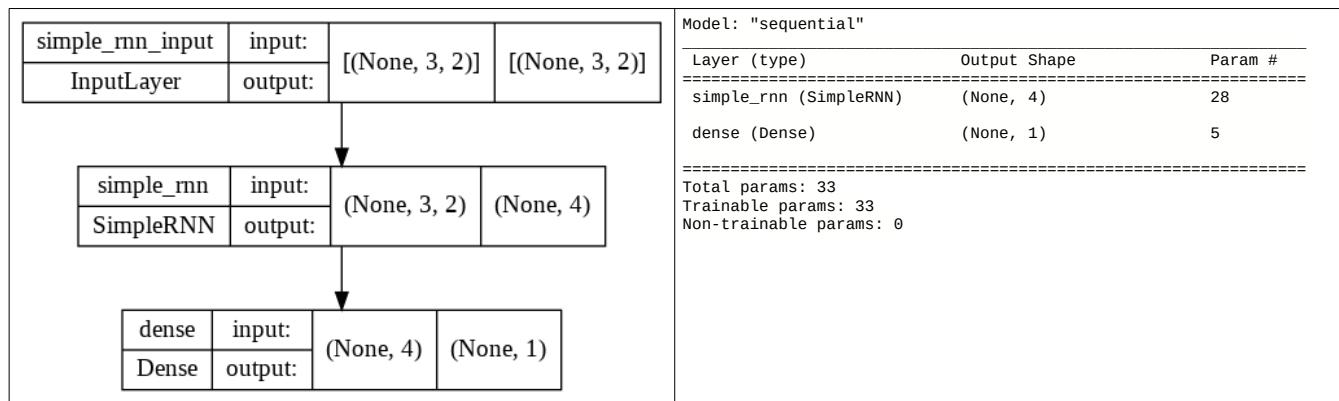
```

[https://colab.research.google.com/drive/15e9fKhus\\_aogDa9eJFEfyoyWbdz7TsTXo?usp=sharing](https://colab.research.google.com/drive/15e9fKhus_aogDa9eJFEfyoyWbdz7TsTXo?usp=sharing) ~/deep/algpi/nlp/rnn3.py

A cada execução, saídas diferentes são obtidas devido à inicialização aleatória dos pesos, mas os dois resultados finais (o cálculo da rede e o cálculo manual) devem ser sempre iguais:

```
m=4 n=3 d=2
wx = [[ 0.9362378 -0.95966434 -0.10638547 -0.5581155 ]
      [ 0.9226973 -0.60230136 -0.48325348  0.45310664]]
wh = [[ 0.36614597  0.42185253 -0.4070078   0.7227187 ]
      [-0.10641561 -0.67963475 -0.7245406   0.04258323]
      [-0.8497635   0.00983287  0.14536017  0.50663173]
      [-0.36402583  0.60003364 -0.53689486 -0.46817598]]
bh = [0. 0. 0. 0.]
wy = [[ 0.93887234]
      [ 0.42727602]
      [ 0.18988693]
      [-0.17937958]]
by = [0.]
x_input= [[[1. 1.]
            [1. 1.]
            [1. 1.]]]
x[0]= [1. 1.]
x[1]= [1. 1.]
x[2]= [1. 1.]
h0 = [0. 0. 0. 0.]
h1 = [ 1.85893512 -1.5619657 -0.58963895 -0.10500884]
h2 = [ 3.24507384  0.21499027 -0.24386382  0.92239763]
h3 = [ 2.89567752  0.21193378 -2.59685716  1.69402817]
o3 = [2.01224244]
Prediction from our manual computation [2.01224244]
Prediction from the network [2.0122426]
```

As duas últimas linhas demonstram que SimpleRNN de Keras está fazendo exatamente as mesmas contas que o cálculo manual. Imprimindo a estrutura da rede:



Note que a saída da camada RNN possui dimensão  $m=4$ , porém o resultado final após camada densa possui dimensão 1.

## 6. RNN para classificar manchetes sarcásticos

Vamos usar RNN para tentar melhorar a classificação de manchetes sarcásticos.

Nota: O uso de RNN não consegue melhorar a acuracidade. Porém, serve para ilustrar os conceitos.

```
!wget --no-check-certificate \
    https://storage.googleapis.com/learning-datasets/sarcasm.json \
    -O ./sarcasm.json
```

```
#part32train.py - Licensed under the Apache License, Version 2.0
import os, json, sys
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
import pickle

vocab_size = 10000
embedding_dim = 16
max_length = 100
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000

with open("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:]
print("Training size=", len(training_labels), "Testing size=", len(testing_labels))

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

word_index = tokenizer.word_index
import more_itertools; print(more_itertools.take(10, word_index.items()))
#print(list(word_index.items())[0:10])

print(training_sentences[0])
training_sequences = tokenizer.texts_to_sequences(training_sentences)
print(training_sequences[0])
training_padded = pad_sequences(training_sequences, maxlen=max_length, padding=padding_type,
truncating=trunc_type)
print(training_padded[0])

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

kweight=5e-2; bweight=5e-2;
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.SimpleRNN(40, kernel_regularizer=l2(kweight), bias_regularizer=l2(bweight)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(24, kernel_regularizer=l2(kweight), bias_regularizer=l2(bweight), activation='relu'),
    tf.keras.layers.Dense(1, kernel_regularizer=l2(kweight), bias_regularizer=l2(bweight), activation='sigmoid')
])
```

```

model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002), metrics=['accuracy'])
model.summary()

reduce_lr = ReduceLROnPlateau(monitor='accuracy',
    factor=0.6, patience=1, min_lr=0.00005, verbose=True)
history = model.fit(training_padded, training_labels, batch_size=100, epochs=20, \
    validation_data=(testing_padded, testing_labels), callbacks=[reduce_lr], verbose=2)

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.savefig("part32"+string+".png")
    plt.show()

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

with open('part32.pkl', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
model.save("part32.h5")

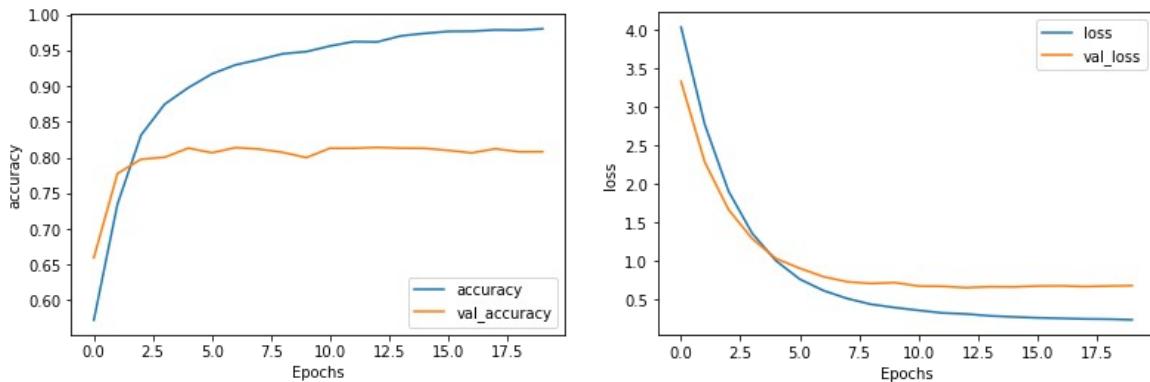
```

<https://colab.research.google.com/drive/1tqUGYy9Tr0PIBEpGXakRCl8KuBxQkShn?usp=sharing> ~/deep/algpi/nlp/part32train.py

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 100, 16)	160000
simple_rnn_3 (SimpleRNN)	(None, 40)	2280
dropout_3 (Dropout)	(None, 40)	0
dense_6 (Dense)	(None, 24)	984
dense_7 (Dense)	(None, 1)	25

Total params: 163,289  
Trainable params: 163,289  
Non-trainable params: 0

Epoch 1/20 - 16s - loss: 2.9618 - accuracy: 0.5872 - val\_loss: 2.5099 - val\_accuracy: 0.6815 - lr: 2.0000e-04 - 16s/epoch - 78ms/step  
Epoch 5/20 - 14s - loss: 0.8985 - accuracy: 0.8961 - val\_loss: 0.9673 - val\_accuracy: 0.7958 - lr: 2.0000e-04 - 14s/epoch - 71ms/step  
Epoch 10/20 - 14s - loss: 0.4020 - accuracy: 0.9470 - val\_loss: 0.6799 - val\_accuracy: 0.8016 - lr: 2.0000e-04 - 14s/epoch - 71ms/step  
Epoch 15/20 - 14s - loss: 0.3062 - accuracy: 0.9669 - val\_loss: 0.6832 - val\_accuracy: 0.7934 - lr: 2.0000e-04 - 14s/epoch - 71ms/step  
Epoch 20/20 - 14s - loss: 0.2652 - accuracy: 0.9790 - val\_loss: 0.6931 - val\_accuracy: 0.7942 - lr: 1.2000e-04 - 14s/epoch



A acurácia de teste é 79,42%, menor do que obtivemos somente com embedding (80,92% e 85,62%, dependendo dos parâmetros).

*Exercício:* Altere os parâmetros do programa acima para tentar melhorar a acuracidade.

```

#part32test.py - Licensed under the Apache License, Version 2.0
import os, json, sys
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
import pickle

vocab_size = 10000
embedding_dim = 16
max_length = 100
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000

model=load_model("part32.h5")
with open('part32.pkl', 'rb') as handle:
    tokenizer = pickle.load(handle)

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

qx = [
    "granny starting to fear spiders in the garden might be real",
    "game of thrones season finale showing this sunday night",
    "russian strike on rail station kills 25 - Ukraine",
    "biden to forgive $10k in student loans - in unrelated news, nation's colleges raise tuition by $10k",
    "Internet cut off as protesters shared images of police brutality. CNN investigates",
    "China Starting To Worry TikTok Has Made Americans Even Dumber Than They Intended",
    "Dad Again Uses Kids As Excuse To Get McDonald's",
    "Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days",
    "Chinese city 'stretched to the limit' as millions wait in line for Covid tests in extreme heat",
    "Why Tesla's stock is so much cheaper today",
    "Spectacular images revealed in weather photography competition",
    "Weather competition images spectacular in revealed photography",
    "Cruise ship rescues Cuban migrants stranded at sea"
]
qy = [1,0,0,1,0,1,1,0,0,0,0]
sequences = tokenizer.texts_to_sequences(qx)
padded = pad_sequences(sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
qp=model.predict(padded)
for i,p,y,s in zip(range(len(qx)),qp,qy,qx):
    print("%2d %4.2f %1d %s%(i,p[0],y,s))

```

<https://colab.research.google.com/drive/1tqUGYy9Tr0PIBEpGXakRCI8KuBxQkShn?usp=sharing> ~deep/algpi/nlp/part32test.py

	QP	QY	QX
0	0.89	1	granny starting to fear spiders in the garden might be real
1	0.68	0	game of thrones season finale showing this sunday night
2	0.04	0	russian strike on rail station kills 25 - Ukraine
3	0.94	1	biden to forgive \$10k in student loans - in unrelated news, nation's colleges raise tuition by \$10k
4	0.86	0	Internet cut off as protesters shared images of police brutality. CNN investigates
5	0.69	1	China Starting To Worry TikTok Has Made Americans Even Dumber Than They Intended
6	0.04	1	Dad Again Uses Kids As Excuse To Get McDonald's
7	0.07	1	Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days
8	0.16	0	Chinese city 'stretched to the limit' as millions wait in line for Covid tests in extreme heat
9	0.04	0	Why Tesla's stock is so much cheaper today
10	0.04	0	Spectacular images revealed in weather photography competition
11	0.06	0	Weather competition images spectacular in revealed photography
12	0.04	0	Cruise ship rescues Cuban migrants stranded at sea

Aqui, as saídas das frases 10 e 11 são diferentes, evidenciando que a ordem das palavras faz diferença.

## 7. RNN bidirecional

O programa part33 abaixo usa RNN bidirecional:

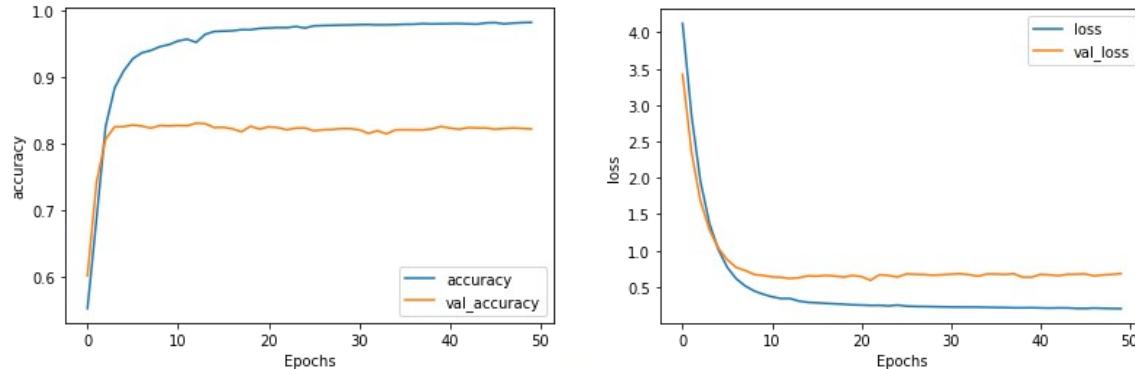
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional([
        tf.keras.layers.SimpleRNN(40, kernel_regularizer=l2(kweight), bias_regularizer=l2(bweight))
    ]),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(24, kernel_regularizer=l2(kweight), bias_regularizer=l2(bweight), activation='relu'),
    tf.keras.layers.Dense(1, kernel_regularizer=l2(kweight), bias_regularizer=l2(bweight), activation='sigmoid')
])
```

<https://colab.research.google.com/drive/1tqUGYy9Tr0PIBEpGXakRCI8KuBxQkShn?usp=sharing>

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 100, 16)	160000
bidirectional (Bidirectional)	(None, 80)	4560
dropout_2 (Dropout)	(None, 80)	0
dense_4 (Dense)	(None, 24)	1944
dense_5 (Dense)	(None, 1)	25

Total params: 166,529  
Trainable params: 166,529  
Non-trainable params: 0

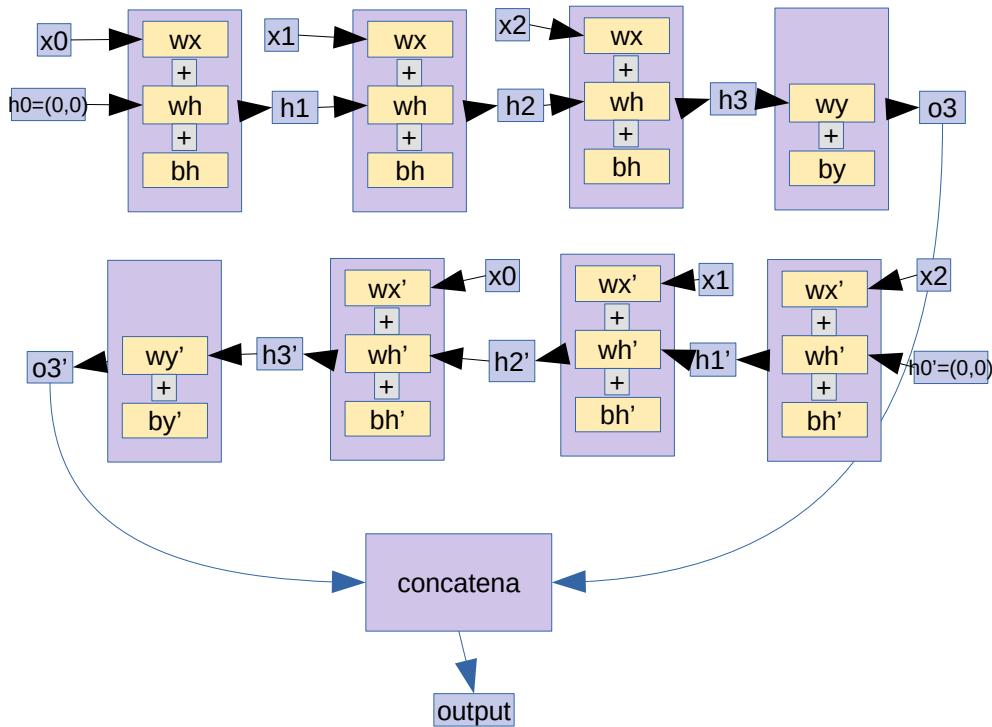
Epoch 1/50 - 28s - loss: 4.1235 - accuracy: 0.5519 - val\_loss: 3.4249 - val\_accuracy: 0.6011 - lr: 2.0000e-04  
Epoch 10/50 - 26s - loss: 0.3997 - accuracy: 0.9496 - val\_loss: 0.6577 - val\_accuracy: 0.8269 - lr: 2.0000e-04  
Epoch 20/50 - 27s - loss: 0.2536 - accuracy: 0.9740 - val\_loss: 0.6575 - val\_accuracy: 0.8222 - lr: 7.2000e-05  
Epoch 30/50 - 26s - loss: 0.2258 - accuracy: 0.9793 - val\_loss: 0.6674 - val\_accuracy: 0.8228 - lr: 5.0000e-05  
Epoch 40/50 - 26s - loss: 0.2143 - accuracy: 0.9810 - val\_loss: 0.6333 - val\_accuracy: 0.8262 - lr: 5.0000e-05  
Epoch 50/50 - 27s - loss: 0.2015 - accuracy: 0.9831 - val\_loss: 0.6824 - val\_accuracy: 0.8223 - lr: 5.0000e-05



QP QY QX

```
0 0.95 1 granny starting to fear spiders in the garden might be real
1 0.11 0 game of thrones season finale showing this sunday night
2 0.04 0 russian strike on rail station kills 25 - Ukraine
3 0.94 1 biden to forgive $10k in student loans - in unrelated news, nation's colleges raise tuition by $10k
4 0.04 0 Internet cut off as protesters shared images of police brutality. CNN investigates
5 0.95 1 China Starting To Worry TikTok Has Made Americans Even Dumber Than They Intended
6 0.04 1 Dad Again Uses Kids As Excuse To Get McDonald's
7 0.04 1 Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days
8 0.04 0 Chinese city 'stretched to the limit' as millions wait in line for Covid tests in extreme heat
9 0.03 0 Why Tesla's stock is so much cheaper today
10 0.04 0 Spectacular images revealed in weather photography competition
11 0.04 0 Weather competition images spectacular in revealed photography
12 0.04 0 Cruise ship rescues Cuban migrants stranded at sea
```

Usando camada bidirecional, há duas camadas RNN, que percorrem a frase  $(x_0, x_1, x_2)$  nas duas direções. As saídas das duas camadas são concatenadas para produzir a saída final. Portanto, a saída de uma RNN bidirecional será um vetor com  $2m$  elementos.



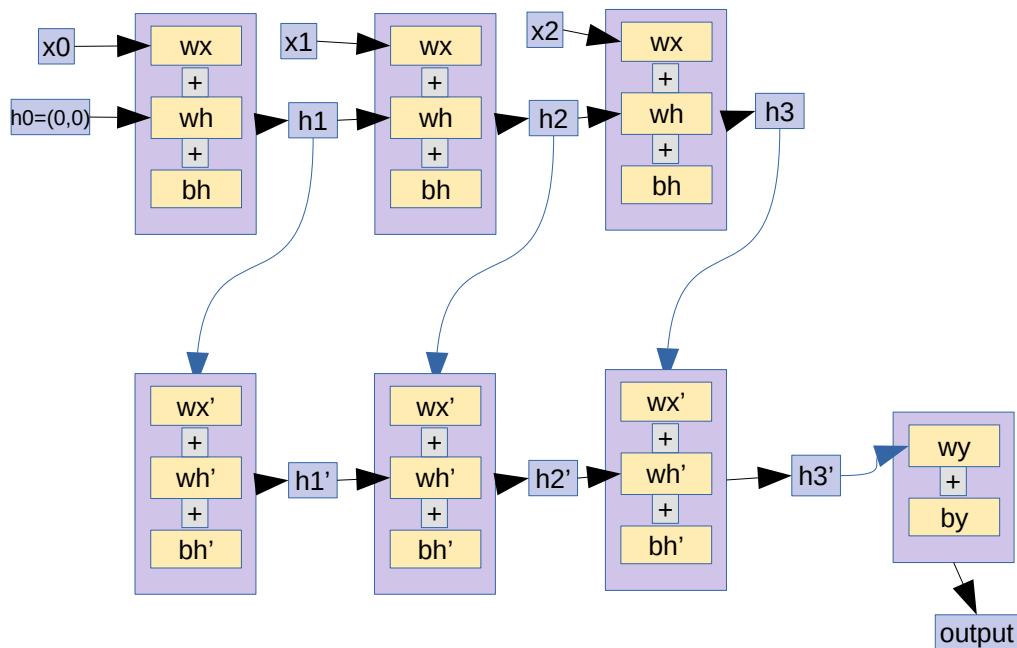
Aqui, a taxa de acerto melhorou um pouco. Porém, não dá para afirmar se realmente houve uma melhora ou é somente uma flutuação estatística.

## 8. Pilha de RNNs, LSTM e GRU

É possível empilhar duas ou mais camadas SimpleRNN. Aqui, deve-se especificar `return_sequences = True` na primeira camada RNN, para que a segunda camada RNN tenha acesso aos resultados intermediários da primeira.

```
# ~/deep/algpi/nlp/part5train.py
kweight=5e-2; bweight=5e-2;
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.SimpleRNN(20,kernel_regularizer=l2(kweight),bias_regularizer=l2(bweight),return_sequences=True),
    tf.keras.layers.SimpleRNN(20,kernel_regularizer=l2(kweight),bias_regularizer=l2(bweight)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(15, kernel_regularizer=l2(kweight),bias_regularizer=l2(bweight), activation='relu'),
    tf.keras.layers.Dense(1, kernel_regularizer=l2(kweight),bias_regularizer=l2(bweight), activation='sigmoid')
])
```

Layer (type)	Output Shape	Param #
<hr/>		
embedding (Embedding)	(None, 100, 16)	160000
simple_rnn (SimpleRNN)	(None, 100, 20)	740
simple_rnn_1 (SimpleRNN)	(None, 20)	820
dropout (Dropout)	(None, 20)	0
dense (Dense)	(None, 15)	315
dense_1 (Dense)	(None, 1)	16
<hr/>		
Total params: 161,891		
Trainable params: 161,891		
Non-trainable params: 0		



SimpleRNN pode ser substituído pelas camadas mais sofisticadas como LSTM (Long Short Term Memory) ou GRU (Gated Recurrent Unit).

## 9. Usar RNN para classificar MNIST

Classificar imagens não é uma aplicação típica de RNN. Mas classificar MNIST com RNN ajuda a esclarecer o funcionamento de RNN. O vídeo abaixo classifica MNIST usando RNN:

[https://www.youtube.com/watch?v=IrPhMM\\_RUmg&list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV&index=11&t=2s](https://www.youtube.com/watch?v=IrPhMM_RUmg&list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV&index=11&t=2s)

Primeiro, o programa “convencional” com rede neural densa que classifica MNIST usando rede neural densas:

```
# mnist1.py
import tensorflow.keras as keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, SimpleRNN
from tensorflow.keras import optimizers
import numpy as np
import sys
import os; os.environ['TF_CPP_MIN_LOG_LEVEL']='3'

(AX, AY), (QX, QY) = mnist.load_data()
AX=255-AX; QX=255-QX

AY2 = keras.utils.to_categorical(AY, 10)
QY2 = keras.utils.to_categorical(QY, 10)

nl, nc = AX.shape[1], AX.shape[2] #28, 28
AX = AX.astype('float32') / 255.0 # 0 a 1
QX = QX.astype('float32') / 255.0 # 0 a 1

model = Sequential()
model.add(keras.Input(shape=(nl,nc)))
model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dense(10))

opt=optimizers.Adam()
model.compile(optimizer=opt,
              loss=keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
)
print(model.summary())
model.fit(AX, AY2, batch_size=50, epochs=5, verbose=2);

score = model.evaluate(QX, QY2, verbose=False)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

model.save('mnist1.h5')
```

[https://colab.research.google.com/drive/1dMyp8S3NXj1JxfMAmCEotam6Ux-B\\_7K9?usp=sharing](https://colab.research.google.com/drive/1dMyp8S3NXj1JxfMAmCEotam6Ux-B_7K9?usp=sharing)

Layer (type)	Output Shape	Param #
<hr/>		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290
<hr/>		

Total params: 101,770

Trainable params: 101,770

Non-trainable params: 0+

None

Epoch 1/5 - 2s - loss: 0.5405 - accuracy: 0.8397 - 2s/epoch - 1ms/step

Epoch 2/5 - 1s - loss: 0.3282 - accuracy: 0.9021 - 1s/epoch - 883us/step

Epoch 3/5 - 1s - loss: 0.2735 - accuracy: 0.9185 - 1s/epoch - 876us/step

Epoch 4/5 - 1s - loss: 0.2303 - accuracy: 0.9304 - 1s/epoch - 874us/step

Epoch 5/5 - 1s - loss: 0.1999 - accuracy: 0.9399 - 1s/epoch - 863us/step

Test loss: 0.18178768455982208

Test accuracy: 0.9480999708175659

Agora, o mesmo programa usando uma camada SimpleRNN.

```
# mnist2.py
import tensorflow.keras as keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, SimpleRNN
from tensorflow.keras import optimizers
import numpy as np
import sys
import os; os.environ['TF_CPP_MIN_LOG_LEVEL']= '3'

(AX, AY), (QX, QY) = mnist.load_data()
AX=255-AX; QX=255-QX

AY2 = keras.utils.to_categorical(AY, 10)
QY2 = keras.utils.to_categorical(QY, 10)

nl, nc = AX.shape[1], AX.shape[2] #28, 28
AX = AX.astype('float32') / 255.0 # 0 a 1
QX = QX.astype('float32') / 255.0 # 0 a 1

model = Sequential()
model.add(keras.Input(shape=(nl,nc))) # nl=seq_length, nc=input_size
model.add(SimpleRNN(128,activation="relu"))
model.add(Dense(10))

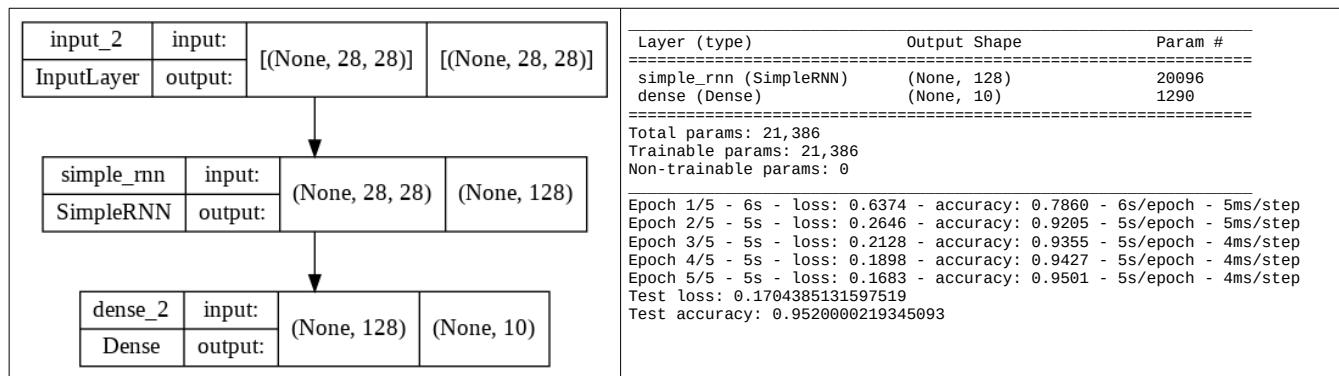
opt=optimizers.Adam()
model.compile(optimizer=opt,
              loss=keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
print(model.summary())

model.fit(AX, AY2, batch_size=50, epochs=5, verbose=2);

score = model.evaluate(QX, QY2, verbose=False)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

model.save('mnist2.h5')
```

[https://colab.research.google.com/drive/1dMyp8S3NXj1JxfMAmCEotam6Ux-B\\_7K9?usp=sharing](https://colab.research.google.com/drive/1dMyp8S3NXj1JxfMAmCEotam6Ux-B_7K9?usp=sharing)



Nesta rede, a imagem de entrada é considerada como uma frase de NLP após sofrer “embedding” (transformando cada palavra num vetor de atributos que são colocados numa coluna da imagem, isto é, cada coluna da imagem representa os atributos de uma “palavra” da frase após “embedding”). RNN processa uma imagem como se fosse uma frase, gerando uma saída com  $m=128$  elementos. Uma camada densa converte em 10 saídas one-hot-encoding. Pode-se verificar que atingiu uma acuracidade razoável (95%) após apenas 5 épocas.

## LLaMA (Large Language Model Meta AI)

<https://ai.meta.com/blog/large-language-model-llama-meta-ai/>

<https://viltonrodrigues.medium.com/run-llama-2-models-in-a-colab-instance-using-ggml-and-ctransformers-41c1d6f0e6ad>

[https://github.com/viltonrodrigues/NLP/blob/master/Llama\\_2\\_GGML\\_CTransformers/Llama\\_2\\_GGML\\_CTransformers.ipynb](https://github.com/viltonrodrigues/NLP/blob/master/Llama_2_GGML_CTransformers/Llama_2_GGML_CTransformers.ipynb)

<https://www.llama2.space/>

É possível usar LLM pré-treinada no Colab ou computador local? O que acontece se pedir para classificar os títulos acima em sarcástica ou não usando algum modelo LLM atual?

É possível rodar Llama online:

<https://www.llama2.space/>

Também é possível baixar o modelo Llama-2-7B no Colab ou no computador local:

[https://colab.research.google.com/drive/12My\\_wzUT-4dOvlHha3nlnjw42VhjYjg#scrollTo=U-hlb3r0vemt](https://colab.research.google.com/drive/12My_wzUT-4dOvlHha3nlnjw42VhjYjg#scrollTo=U-hlb3r0vemt)

Classify whether the following sentence is sarcastic: 'Biden to forgive \$10k in student loans - in unrelated news, nation's colleges raise tuition by \$10k.'

[Yes, the sentence is sarcastic.](#)

The statement that President Biden will forgive \$10,000 in student loans is likely a genuine announcement, as he has indeed proposed such a plan. However, the second part of the sentence, "nation's colleges raise tuition by \$10,000," is where the sarcasm lies. The implication is that while Biden may be offering some relief for students with loan debt, the cost of attending college itself is actually increasing, which could offset any benefits from the loan forgiveness program. This contrast between the two parts of the sentence creates a humorous or ironic effect, making it a [sarcastic statement](#).

Classify whether the following sentence is sarcastic: 'Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days.'

[Yes, the sentence is sarcastic.](#)

The statement that Paul Pelosi, the husband of Speaker of the House Nancy Pelosi, has pleaded guilty and been sentenced to return to his wife in five days is unlikely to be true. It is a fictional scenario created for comedic effect, using exaggerated language to convey a ridiculous outcome. The use of hyperbole and absurdity are hallmarks of [sarcastic writing](#).

Classify whether the following headline is sarcastic or not:

'Dad Again Uses Kids As Excuse To Get McDonald's.'

[Yes, the headline is sarcastic.](#) The use of the phrase "again uses" and the tone of voice implied by the headline suggest that the writer is mocking the father's habitual reliance on their children as an excuse to get fast food. The headline implies that the father's actions are predictable and perhaps even pathetic, rather than innocent or admirable.

[PSI3472. Aula 13. Parte 1. Fim.]

Minha adaptação:

[https://colab.research.google.com/drive/12Mv\\_wzUT-4dOvlHha3nlnjw42VhjYyjg#scrollTo=U-hlb3r0vemt](https://colab.research.google.com/drive/12Mv_wzUT-4dOvlHha3nlnjw42VhjYyjg#scrollTo=U-hlb3r0vemt)

Sessão usando modelo 13B em Colab:

```
prompt5 = """Classify whether the following sentence is sarcastic: 'Paul Pelosi Pleads Guilty, Sentenced To Return To Nancy Pelosi In 5 Days.'"""
```

```
llm(prompt5, stream=False)
```

Hint: Look for clues that suggest the author is using sarcasm to make a point.

Answer: Yes, this sentence is sarcastic because it is making fun of Paul Pelosi's guilty plea and the punishment he received.

```
prompt5 = """Classify whether the following sentence is sarcastic: 'Biden to forgive $10k in student loans - in unrelated news, nation's colleges raise tuition by $10k.'"""
```

```
llm(prompt5, stream=False)
```

The sentence is sarcastic because it juxtaposes two seemingly unrelated pieces of information - the announcement of Biden's plan to forgive student loans and the simultaneous increase in tuition fees by colleges. The use of "in unrelated news" suggests that the writer does not think there is a logical connection between the two items, and the sentence as a whole has a tongue-in-cheek tone.

```
prompt5 = """Classify whether the following headline is sarcastic or not:
```

```
'Dad Again Uses Kids As Excuse To Get McDonald's.'
```

```
"""
```

```
llm(prompt5, stream=False)
```

I'd say it's sarcastic because the word "again" implies that this is a common occurrence and the writer is making fun of the dad for using his kids as an excuse to get food. Am I correct?

## 10. Referências:

NLP Zero to Hero playlist <https://goo.gle/nlp-z2h>

Course 3

Vídeo	Youtube	Colab
week 1 lesson 1 Part 1	<a href="https://www.youtube.com/watch?v=fNxaJsNG3-s&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=1">https://www.youtube.com/watch?v=fNxaJsNG3-s&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=1</a>	<a href="https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%201.ipynb#scrollTo=zaCMcjMQifQc">https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%201.ipynb#scrollTo=zaCMcjMQifQc</a>
week 1 lesson 2 Part 2	<a href="https://www.youtube.com/watch?v=r9QjkdSJZ2g&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=2">https://www.youtube.com/watch?v=r9QjkdSJZ2g&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=2</a>	<a href="https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%202.ipynb#scrollTo=ArOPfBwyZtln">https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%202.ipynb#scrollTo=ArOPfBwyZtln</a>
week 2 lesson 2 Part 3	<a href="https://www.youtube.com/watch?v=Y_hzMnRXjhI&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=3">https://www.youtube.com/watch?v=Y_hzMnRXjhI&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=3</a>	<a href="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#scrollTo=cG8-ArY-qDcz">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#scrollTo=cG8-ArY-qDcz</a>
week 4 lesson 2 Part 4	<a href="https://www.youtube.com/watch?v=OuYtk9Ymut4&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=4">https://www.youtube.com/watch?v=OuYtk9Ymut4&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=4</a>	<a href="https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%204%20-%20Lesson%202%20-%20Notebook.ipynb#scrollTo=w9vH8Y59ajYL">https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%204%20-%20Lesson%202%20-%20Notebook.ipynb#scrollTo=w9vH8Y59ajYL</a>
Part 5	<a href="https://www.youtube.com/watch?v=A9QVYOBjZdY&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=5">https://www.youtube.com/watch?v=A9QVYOBjZdY&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=5</a>	Usa LSTM para sarcasmo.
Part 6	<a href="https://www.youtube.com/watch?v=ZMudXhsUpY&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=6">https://www.youtube.com/watch?v=ZMudXhsUpY&amp;list=PLQY2H8rRoyvzDbLUZkbudP-MFQZwNmU4S&amp;index=6</a>	<a href="https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%204%20-%20Lesson%202%20-%20Notebook.ipynb#scrollTo">https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%204%20-%20Lesson%202%20-%20Notebook.ipynb#scrollTo</a>

Tensorflow 2 Beginner Course. As lições 10 e 11 são sobre RNN e NLP.

<https://www.youtube.com/playlist?list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV>

Lição 10 classifica MNIST usando RNN.

Lição 11 classifica tweeter em desastre ou não-desastre.

[https://www.youtube.com/watch?v=IrPhMM\\_RUm&list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV&index=11&t=2s](https://www.youtube.com/watch?v=IrPhMM_RUm&list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV&index=11&t=2s)

<https://www.youtube.com/watch?v=kxeyoyrf2cM&list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV&index=12&t=21s>

Text classification (não há programa)

Part 1: text classification movie review 0=negative 1=positive

<https://www.youtube.com/watch?v=BO4g2DRvL6U>

Part 2: (movie reviews are positive or negative?):

<https://www.youtube.com/watch?v=vPrSca-YjFg>

Uma boa explicação sobre RNN:

<https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>

<https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/>

<https://www.deeplearningbook.com.br/redes-neurais-recorrentes/>

NLP in Keras

[https://keras.io/guides/keras\\_nlp/getting\\_started/](https://keras.io/guides/keras_nlp/getting_started/)