Análisis de Datos y Aprendizaje Máquina con Tensorflow 2.0: Redes neuronales recurrentes

2019/09/30

Estructura RNN

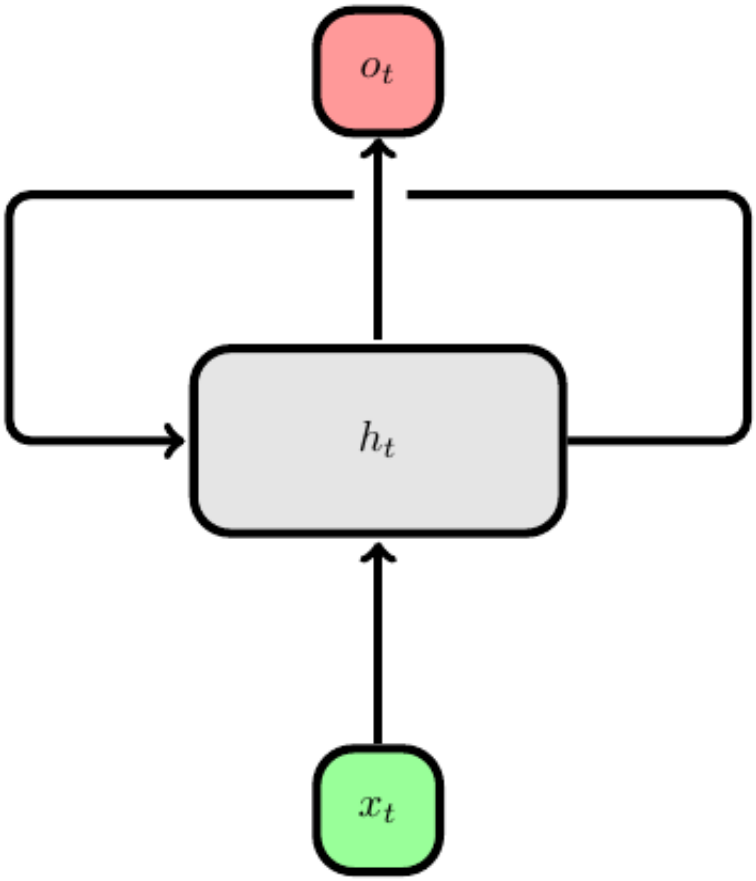
- Objetivo: Conocer la notación de las RNN
- RNN son estructuras que pueden manejar datos con formato "secuencial" al preservar el "estado" anterior
- Muchas RNN trabajan con embeddings en las entradas. Los embeddings son representaciones vectoriales aprendidas por la red, lo cual captura la relación de las palabras.
- A diferencia de las Feedforward neural networks, las RNN pueden manejar secuencias de datos

In [8]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import LSTM, Embedding, Dense, SimpleRNN
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
```

Reseñas de películas de IMDB

- Conjunto de datos de 25,000 críticas de películas de IMDB, etiquetadas por sentimiento (positivo / negativo).

In [9]:
```
# numero de palabras
num_words = 10000

maxlen = 59


(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)


x_train = pad_sequences(x_train, maxlen=maxlen)
```



RNN

```
        x_test = pad_sequences(x_test, maxlen=maxlen)


        print(x_train.shape)
        print(x_test.shape)
        print(y_train.shape)
        print(y_test.shape)
```

```
(25000, 59)
(25000, 59)
(25000,)
(25000,)
```

```
In [10]: epoch = 10
         verbose = 1
         batch = 30
```

Cada palabra de la review esta identificada por un número

```
In [11]: print('Reseña')
         print(x_train[0])
         print('Etiqueta')
         print(y_train[0])
```

```
Reseña
[  13  104   88    4  381   15  297   98   32 2071   56   26  141    6
  194 7486   18    4  226   22   21  134  476   26  480    5  144   30
 5535   18   51   36   28  224   92   25  104    4  226   65   16   38
 1334   88   12   16  283    5   16 4472  113  103   32   15   16 5345
   19  178   32]
Etiqueta
1
```

Palabras de reseña

```
In [12]: wordDict = {y:x for x,y in imdb.get_word_index().items()}
         res = []
         for index in x_train[0]:
             res.append(wordDict.get(index - 3))
         print('Reseña: ',res,'Longitud reseña: ', len(res))
```

```
Reseña:  ['i', 'think', 'because', 'the', 'stars', 'that', 'play', 'them', 'all', 'grown', 'up',
```

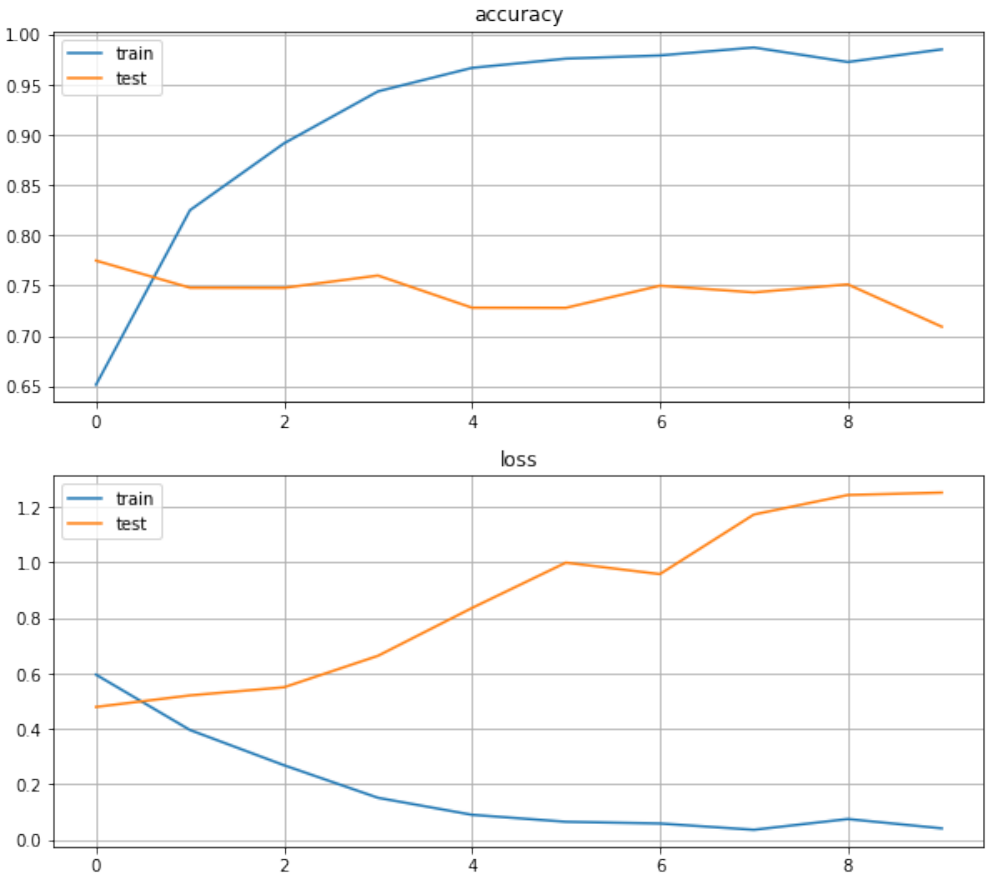
RNN simple con embedding

- Los embeddings son representaciones vectoriales de palabras o caracteres

```
In [13]: model = Sequential()
         model.add(Embedding(num_words, 128))
         model.add(SimpleRNN(128))
         model.add(Dense(1, activation='sigmoid'))

In [14]: model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         model.summary()

         history = model.fit(x_train, y_train, batch_size=batch, epochs=epoch, verbose = verbose,
                             validation_split = 0.3)

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 128)         1280000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 128)               32896
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,313,025
Trainable params: 1,313,025
Non-trainable params: 0
_________________________________________________________________
Train on 17500 samples, validate on 7500 samples
Epoch 1/10
17500/17500 [==============================] - 24s 1ms/sample - loss: 0.5952 - accuracy: 0.6515 -
Epoch 2/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.3958 - accuracy: 0.8251 -
Epoch 3/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.2690 - accuracy: 0.8916 -
Epoch 4/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.1517 - accuracy: 0.9433 -
Epoch 5/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.0904 - accuracy: 0.9667 -
Epoch 6/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.0652 - accuracy: 0.9759 -
Epoch 7/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.0591 - accuracy: 0.9790 -
Epoch 8/10
17500/17500 [==============================] - 22s 1ms/sample - loss: 0.0364 - accuracy: 0.9870 -
Epoch 9/10
17500/17500 [==============================] - 22s 1ms/sample - loss: 0.0751 - accuracy: 0.9725 -
Epoch 10/10
17500/17500 [==============================] - 23s 1ms/sample - loss: 0.0415 - accuracy: 0.9851 -

In [15]: test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

         print('\nTest acccuracy:', test_acc)
```

```
25000/1 - 5s - loss: 1.3845 - accuracy: 0.7146

Test acccuracy: 0.71464
```

In [16]:
```
plt.figure(figsize=(10,9))

plt.subplot(211)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('accuracy')
plt.legend(['train', 'test'])
plt.grid()
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss')
plt.legend(['train', 'test'])
plt.grid()
```

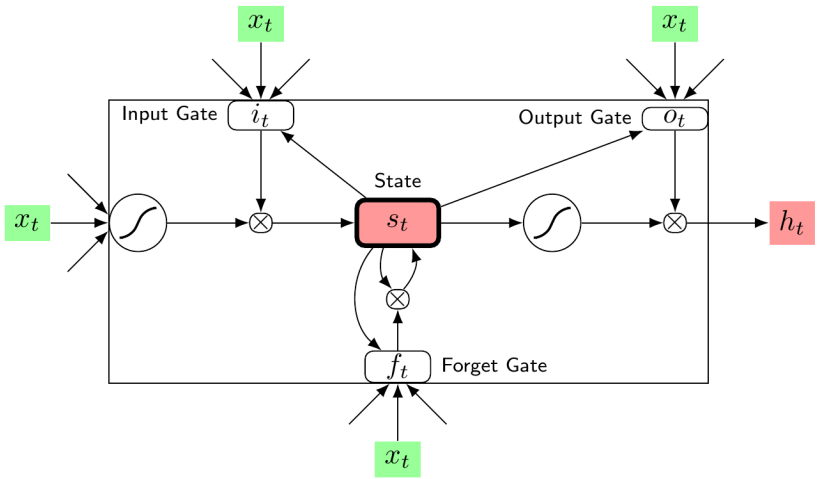


```
In [17]: test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

         print('\nTest acccuracy:', test_acc)
```

```
25000/1 - 5s - loss: 1.3845 - accuracy: 0.7146

Test acccuracy: 0.71464
```

LSTM

- En LSTM se toma en cuenta el estado *st*, el cual sirve para olvidar o recordar parámetros relevantes
- Las LSTM cuentan con más matrices, por lo que tienen más parámetros que una RNN normal, lo que hace más lento su entrenamiento




```
In [18]: model = Sequential()
         model.add(Embedding(num_words, 128))
         model.add(LSTM(128))
         model.add(Dense(1, activation='sigmoid'))
```

```
In [19]: model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         model.summary()

         history = model.fit(x_train, y_train, batch_size=batch, epochs=epoch, verbose = verbose,
                             validation_split = 0.3)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 128)         1280000
_________________________________________________________________
```

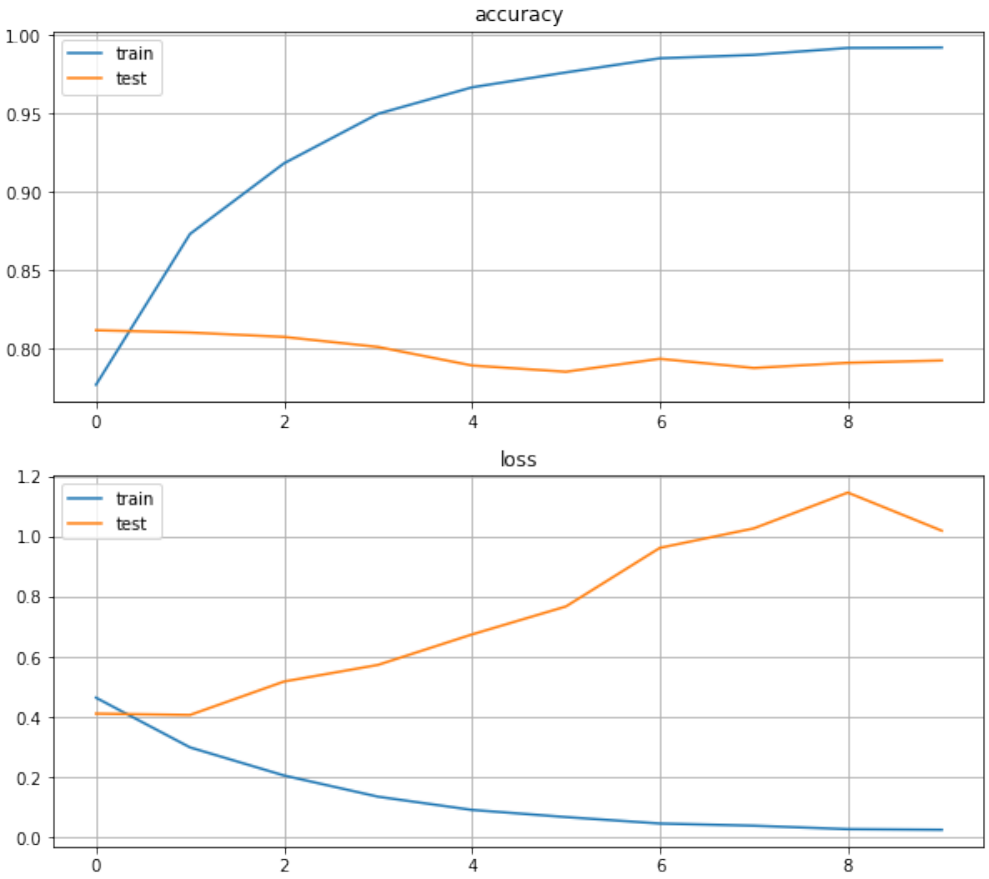
```
lstm (LSTM)                  (None, 128)               131584
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,411,713
Trainable params: 1,411,713
Non-trainable params: 0
_________________________________________________________________
Train on 17500 samples, validate on 7500 samples
Epoch 1/10
17500/17500 [==============================] - 14s 790us/sample - loss: 0.4632 - accuracy: 0.7768
Epoch 2/10
17500/17500 [==============================] - 12s 667us/sample - loss: 0.2979 - accuracy: 0.8731
Epoch 3/10
17500/17500 [==============================] - 12s 664us/sample - loss: 0.2044 - accuracy: 0.9183
Epoch 4/10
17500/17500 [==============================] - 12s 670us/sample - loss: 0.1339 - accuracy: 0.9499
Epoch 5/10
17500/17500 [==============================] - 12s 668us/sample - loss: 0.0898 - accuracy: 0.9669
Epoch 6/10
17500/17500 [==============================] - 12s 670us/sample - loss: 0.0660 - accuracy: 0.9764
Epoch 7/10
17500/17500 [==============================] - 13s 737us/sample - loss: 0.0445 - accuracy: 0.9854
Epoch 8/10
17500/17500 [==============================] - 14s 806us/sample - loss: 0.0372 - accuracy: 0.9876
Epoch 9/10
17500/17500 [==============================] - 13s 743us/sample - loss: 0.0258 - accuracy: 0.9921
Epoch 10/10
17500/17500 [==============================] - 13s 723us/sample - loss: 0.0237 - accuracy: 0.9923
```

```
In [20]: test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

         print('\nTest acccuracy:', test_acc)
```

```
25000/1 - 3s - loss: 1.1409 - accuracy: 0.7889

Test acccuracy: 0.78892
```

```
In [21]: plt.figure(figsize=(10,9))

         plt.subplot(211)
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('accuracy')
         plt.legend(['train', 'test'])
         plt.grid()
         plt.subplot(212)
         plt.plot(history.history['loss'])
```

```
        plt.plot(history.history['val_loss'])
        plt.title('loss')
        plt.legend(['train', 'test'])
        plt.grid()
```




```
In [22]: test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

         print('\nTest acccuracy:', test_acc)
```

```
25000/1 - 3s - loss: 1.1409 - accuracy: 0.7889

Test acccuracy: 0.78892
```

- Experimentar con diferentes arquitecturas y número de neuronas
- Experimentar con otro dataset
- Experimentar con otros optimizadores y funciones de activación