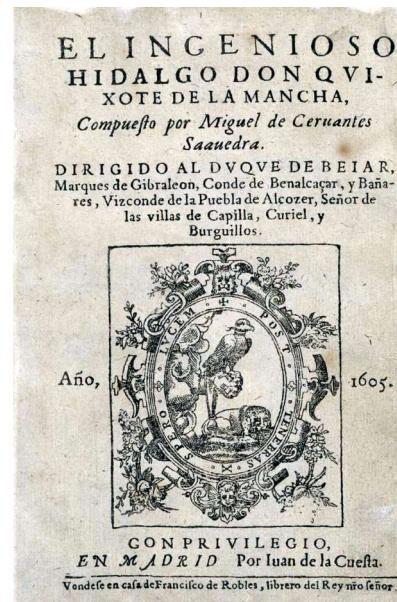


▼ Laboratorio: Modelos del lenguaje con RNNs

En este laboratorio, vamos a entrenar un modelo del lenguaje basado en caracteres con Recurrente modelo para generar texto. En particular, alimentaremos nuestro modelo con obras de la literatura neuronal que sea capaz de "escribir" fragmentos literarios.

Los entrenamientos en esta laboratorio para obtener un modelo de calidad podrían tomar cierto tiempo, aconseja empezar a trabajar pronto. El uso de GPUs no ayuda tanto con LSTMs como con CNNs, cosa es posible que podáis entrenar más rápido o a la misma velocidad que en Colab. En todo caso, para completar este laboratorio con éxito.



El dataset a utilizar consistirá en un archivo de texto con el contenido íntegro en castellano antiguo de 'El Quijote' de la Mancha, disponible de manera libre en la página de [Project Gutenberg](#). Asimismo, como apartadoc, se puede utilizar otras fuentes de texto. Aquí podéis descargar los datos a utilizar de 'El Quijote' y un par de

[El ingenioso hidalgo Don Quijote de la Mancha \(Miguel de Cervantes\)](#)

[Compilación de obras teatrales \(Calderón de la Barca\)](#)

[Trafalgar \(Benito Pérez Galdós\)](#)

Como ya deberíamos de estar acostumbrados en problemas de Machine Learning, es importante comenzar.

▼ 1. Carga y procesado del texto

Primero, vamos a descargar el libro e inspeccionar los datos. El fichero a descargar es una versión en PDF que ha borrado introducciones, licencias y otras secciones para dejarlo con el contenido real de la novela.

```
import numpy as np
import keras
import matplotlib.pyplot as plt
```

https://colab.research.google.com/drive/1qvHNCGiGLBLS4rW-mWtiCpqnxDChodj1#scrollTo=zV7POk_lzEZx&printMode=true

```
from keras.callbacks import LambdaCallback
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU, Flatten
from keras.layers import Dense, Dropout, BatchNormalization, Conv1D, Embedding
import random
import io

path = keras.utils.get_file(
    fname="don_quijote.txt",
    origin="https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resid=C506CF0A4F373B0F"
)
```

↳ Using TensorFlow backend.

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.

We recommend you [upgrade](#) now or ensure your notebook will continue to use TensorFlow 1.x via the %t

```
Downloading data from https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resid=2154496/2151176 [=====] - 2s 1us/step
```

Una vez descargado, vamos a leer el contenido del fichero en una variable. Adicionalmente, convierte el texto para ponérselo un poco más fácil a nuestro modelo (de modo que todas las letras sean minúsculas y no mezclen minúsculas y mayúsculas).

1.1. Leer todo el contenido del fichero en una única variable `text` y convertir el string a minúsculas

```
# Cargamos el fichero
file = open(path)
text = file.read()
text = text.lower() # convertimos a minuscula
#print(text)
origtext = text
```

Podemos comprobar ahora que efectivamente nuestra variable contiene el resultado deseado, con lo siguiente:

```
print("Longitud del texto: {}".format(len(text)))
print(text[0:300])
```

↳ Longitud del texto: 2071198

capítulo primero. que trata de la condición y ejercicio del famoso hidalgo don quijote de la mancha

en un lugar de la mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor. una olla de algo más

▼ 2. Procesado de los datos

Una de las grandes ventajas de trabajar con modelos que utilizan caracteres en vez de palabras es la posibilidad de tratar el texto de forma más sencilla (partirlo palabra a palabra). Nuestro modelo funcionará directamente con los caracteres en el texto.

Antes de hacer nada, necesitamos procesar el texto en entradas y salidas compatibles con nuestro lenguaje con RNNs acepta una serie de caracteres y predice el siguiente carácter en la secuencia

- "El ingenioso don Qui" -> predicción: **j**
- "El ingenioso don Quij" -> predicción: **o**

De modo que la entrada y la salida de nuestro modelo necesita ser algo parecido a este esquema de preparar los datos para nuestro modelo.

1. **Secuencia a secuencia.** La entrada de nuestro modelo sería una secuencia y la salida sería esa secuencia en cada instante de tiempo la RNN tiene que predecir el carácter siguiente. Por ejemplo:

- *Input:* El ingenioso don Quijot
- *Output:* I ingenioso don Quijote

2. **Secuencia a carácter.** En este variante, pasariamos una secuencia de caracteres por nuestra RNN y, al llegar, obtendríamos el siguiente carácter.

- *Input:* El ingenioso don Quijot
- *Output:* e

En este laboratorio, por simplicidad, vamos a utilizar la segunda variante.

De este modo, a partir del texto, haremos de generar nuestro propio training data que consista en una secuencia de caracteres y su carácter a predecir. Para estandarizar las cosas, utilizaremos secuencias de tamaño *SEQ_LENGTH* (que nosotros elegiremos nosotros).

▼ 2.1. Obtención de los caracteres y mapas de caracteres

Antes que nada, necesitamos saber qué caracteres aparecen en el texto, ya que tendremos que devolver *num_chars - 1* en el modelo. Obtener:

1. Número de caracteres únicos que aparecen en el texto.
2. Diccionario que asocia char a índice único entre 0 y *num_chars - 1*. Por ejemplo, {'a': 0, 'b': 1, ...}
3. Diccionario inverso de índices a caracteres: {0: 'a', 1: 'b', ...}

TU CÓDIGO AQUÍ

```
# 1 Número de caracteres únicos que aparecen en el texto.
uniqchar = list(set(text))
print(uniqchar)

# 2 Diccionario que asocia char a índice único entre 0 y num_chars - 1. Por ejemplo, {'a': 0, 'b': 1, ...}
num_chars = len(uniqchar)
mindex = range(0,num_chars)

dirdict = dict(zip(uniqchar, mindex))
print(dirdict)

# 3 Diccionario inverso de índices a caracteres: {0: 'a', 1: 'b', ...}
```

```
invdict = dict(zip(mindex,uniqchar))
print(invdict)
```

```
↳ ['l', 'ü', 'q', '-', '(', 'g', 'm', '6', 't', 'f', 'ç', 'o', 'a', 'd', 'b', ')', 'í',
{'l': 0, 'ü': 1, 'q': 2, '-': 3, '(': 4, 'g': 5, 'm': 6, '6': 7, 't': 8, 'f': 9, 'ç':
{0: 'l', 1: 'ü', 2: 'q', 3: '-', 4: '(', 5: 'g', 6: 'm', 7: '6', 8: 't', 9: 'f', 10:
```

▼ 2.2. Obtención de secuencias de entrada y carácter a predecir

Ahora, vamos a obtener las secuencias de entrada en formato texto y los correspondientes caracteres completos leídos anteriormente, obteniendo una secuencia de SEQ_LENGTH caracteres y el siguiente desplazarse un carácter a la izquierda y hacer lo mismo para obtener una nueva secuencia y predicción **sequences** y los caracteres a predecir en una variable **next_chars**.

Por ejemplo, si el texto fuera "Don Quijote" y SEQ_LENGTH fuese 5, tendríamos

- **sequences** = ["Don Q", "on Qu", "n Qui", " Quij", "Quijo", "uijot"]
- **next_chars** = ['u', 'i', 'j', 'o', 't', 'e']

```
# Definimos el tamaño de las secuencias. Puedes dejar este valor por defecto.
SEQ_LENGTH = 30
```

```
sequences = []
next_chars = []
tamtext = len(text)

## TU CÓDIGO AQUÍ
for i in range(0,tamtext-SEQ_LENGTH):
    sequences.append(text[i:SEQ_LENGTH+i])
    next_chars.append(text[SEQ_LENGTH+i])

print('Sequences : = ',sequences[0:500])
print('Next chars : = ',next_chars[0:500])
```

```
↳ Sequences : = ['capítulo primero. que trata de', 'apítulo primero. que trata de ', 'Next chars : = [' ', 'l', 'a', ' ', 'c', 'o', 'n', 'd', 'i', 'c', 'i', 'ó', 'n', ' ']
```

Indicar el tamaño del training set que acabamos de generar.

```
print('Tam Sequences : = ',len(sequences))
```

```
↳ Tam Sequences : = 2071168
```

Como el Quijote es muy largo y tenemos muchas secuencias, podríamos encontrar problemas de memoria con ellas. Si estás corriendo esto localmente y tienes problemas de memoria, puedes reducirlo porque, a menos datos, peor calidad del modelo.

```
MAX_SEQUENCES = 500000
```

```

perm = np.random.permutation(len(sequences))
sequences, next_chars = np.array(sequences), np.array(next_chars)
sequences, next_chars = sequences[perm], next_chars[perm]
sequences, next_chars = list(sequences[:MAX_SEQUENCES]), list(next_chars[:MAX_SEQUENCES])

print(len(sequences))

```

↳ 500000

▼ 2.3. Obtención de input X y output y para el modelo

Finalmente, a partir de los datos de entrenamiento que hemos generado vamos a crear los arrays modelo.

Para ello, vamos a utilizar *one-hot encoding* para nuestros caracteres. Por ejemplo, si sólo tuviéramos 4 representaciones serían: (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0) y (0, 0, 0, 1).

De este modo, **X** tendrá shape (*num_sequences, seq_length, num_chars*) e **y** tendrá shape (*num_sequences, num_chars*).

```

NUM_CHARS = num_chars # Tu número de caracteres distintos aquí
NUM_SEQUENCES = len(sequences)
X = np.zeros((NUM_SEQUENCES, SEQ_LENGTH, NUM_CHARS))
y = np.zeros((NUM_SEQUENCES, NUM_CHARS))

## Tu código para llenar X e y aquí. Pista: utilizar el diccionario de
## chars a índices obtenido anteriormente junto con numpy. Por ejemplo,
## si hacemos
##     X[0, 1, char_to_indices['a']] = 1
## estamos diciendo que para la segunda posición de la primera secuencia se
## tiene una 'a'

## TU CÓDIGO AQUÍ

# bucle en las secuencias
for l in range(0,NUM_SEQUENCES):
    lseq = list(sequences[l])
    y[l,dirdict[next_chars[l]]] = 1
    #bucle en los caracteres
    for k in range(0,SEQ_LENGTH):
        X[l, k, dirdict[lseq[k]]] = 1

```

▼ 3. Definición del modelo y entrenamiento

Una vez tenemos ya todo preparado, es hora de definir el modelo. Define un modelo que utilice un modelo puede definirse de una manera más compleja, para empezar debería bastar con una LSTM que predice el siguiente carácter a producir. Adam puede ser una buena elección de optimizador.

Una vez el modelo esté definido, entrénalo un poco para asegurarte de que la loss es decreciente durante el entrenamiento en el entregable final, ya que vamos a hacer el entrenamiento más informativo en el entregable final.

▼ Modelo 1 - LSTM

```
# Sintesis del modelo
N = 128

model1 = Sequential()
model1.add(LSTM(N,dropout=0.2,recurrent_dropout=0.2,input_shape=(X.shape[1:])))  

model1.add(Dense(int(4*NUM_CHARS),activation='relu'))
model1.add(Dense(NUM_CHARS,activation='softmax'))

model1.name='LSTM'
model1.summary()

# compilamos el modelo
model1.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`
Model: "LSTM"



| Layer (type)    | Output Shape | Param # |
|-----------------|--------------|---------|
| lstm_1 (LSTM)   | (None, 128)  | 97280   |
| dense_1 (Dense) | (None, 244)  | 31476   |
| dense_2 (Dense) | (None, 61)   | 14945   |


Total params: 143,701
Trainable params: 143,701
Non-trainable params: 0

[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:75]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]
[WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python]
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

▼ Modelo 2 GRU

```
# Sintesis del modelo
N = 128

model2 = Sequential()
```

```

model2.add(GRU(N, return_sequences=True,input_shape=(X.shape[1:])))
model2.add(Flatten())
#model1.add(LSTM(N,dropout=0.2,recurrent_dropout=0.2,input_shape=(X.shape[1:]))

model2.add(Dense(int(4*NUM_CHARS),activation='relu'))
model2.add(Dense(NUM_CHARS,activation='softmax'))

model2.name='GRU'
model2.summary()

# compilamos el modelo
model2.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

```

↳ Model: "GRU"

Layer (type)	Output Shape	Param #
<hr/>		
gru_1 (GRU)	(None, 30, 128)	72960
flatten_1 (Flatten)	(None, 3840)	0
dense_3 (Dense)	(None, 244)	937204
dense_4 (Dense)	(None, 61)	14945
<hr/>		
Total params: 1,025,109		
Trainable params: 1,025,109		
Non-trainable params: 0		

▼ Prueba del modelo

```

def plot_compare_alb(vhistory, vname, metric_train, metric_val, title="Graph title"):
    plt.figure(figsize=(12,5))
    for i in range (len(vhistory)):

        plt.plot(vhistory[i].history[metric_train],label='%s Train' % vname[i])

        plt.plot(vhistory[i].history[metric_val],label='%s Test' % vname[i])
    #plt.figure(figsize=(4,5))
    #plt.rcParams["figure.figsize"] = [10,10]
    plt.title(title)
    plt.ylabel(title)
    plt.xlabel('Epoch')
    plt.legend(loc='lower right')
    plt.show()

def plot_compare_alb_loss_acc(vhistory, vname):
    plot_compare_alb(vhistory, vname, 'loss','val_loss', title="Loss comparation")
    plot_compare_alb(vhistory, vname, 'acc', 'val_acc', title="Accuracy comparation")

N = 2000
xtest = X[0:N,:,:]
vtest = v[0:N,:]

```

```
, ----- ,----- ,  
xval = X[N:2*N,:,:]  
yval = y[N:2*N,:]  
  
print(model1.name)  
h1 = model1.fit(xtest,ytest,epochs = 10,verbose= 2,  
                  validation_data = [xval,yval])  
  
print(model2.name)  
h2 = model2.fit(xtest,ytest,epochs = 10,verbose= 2,  
                  validation_data = [xval,yval])
```



```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

Train on 2000 samples, validate on 2000 samples
Epoch 1/10
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorf]

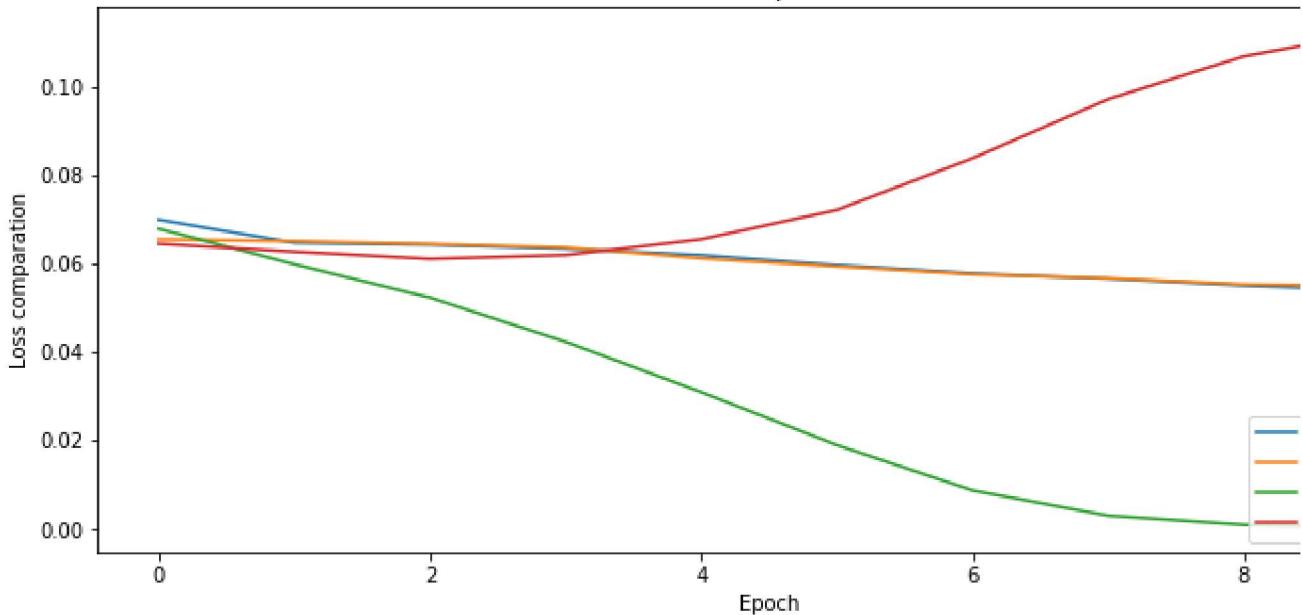
- 9s - loss: 0.0700 - acc: 0.9836 - val_loss: 0.0656 - val_acc: 0.9836
Epoch 2/10
- 4s - loss: 0.0648 - acc: 0.9836 - val_loss: 0.0652 - val_acc: 0.9836
Epoch 3/10
- 4s - loss: 0.0644 - acc: 0.9836 - val_loss: 0.0645 - val_acc: 0.9836
Epoch 4/10
- 4s - loss: 0.0634 - acc: 0.9836 - val_loss: 0.0637 - val_acc: 0.9836
Epoch 5/10
- 4s - loss: 0.0620 - acc: 0.9837 - val_loss: 0.0613 - val_acc: 0.9838
Epoch 6/10
- 4s - loss: 0.0597 - acc: 0.9838 - val_loss: 0.0594 - val_acc: 0.9838
Epoch 7/10
- 4s - loss: 0.0578 - acc: 0.9839 - val_loss: 0.0577 - val_acc: 0.9839
Epoch 8/10
- 4s - loss: 0.0567 - acc: 0.9839 - val_loss: 0.0568 - val_acc: 0.9838
Epoch 9/10
- 4s - loss: 0.0551 - acc: 0.9843 - val_loss: 0.0554 - val_acc: 0.9842
Epoch 10/10
- 4s - loss: 0.0540 - acc: 0.9842 - val_loss: 0.0550 - val_acc: 0.9845
GRU
Train on 2000 samples, validate on 2000 samples
Epoch 1/10
- 4s - loss: 0.0680 - acc: 0.9836 - val_loss: 0.0646 - val_acc: 0.9836
Epoch 2/10
- 3s - loss: 0.0599 - acc: 0.9837 - val_loss: 0.0628 - val_acc: 0.9837
Epoch 3/10
- 3s - loss: 0.0524 - acc: 0.9841 - val_loss: 0.0611 - val_acc: 0.9838
Epoch 4/10
- 3s - loss: 0.0424 - acc: 0.9861 - val_loss: 0.0620 - val_acc: 0.9835
Epoch 5/10
- 3s - loss: 0.0310 - acc: 0.9890 - val_loss: 0.0656 - val_acc: 0.9821
Epoch 6/10
- 3s - loss: 0.0190 - acc: 0.9936 - val_loss: 0.0722 - val_acc: 0.9808
Epoch 7/10
- 3s - loss: 0.0087 - acc: 0.9978 - val_loss: 0.0839 - val_acc: 0.9789
Epoch 8/10
- 3s - loss: 0.0030 - acc: 0.9997 - val_loss: 0.0973 - val_acc: 0.9775
Epoch 9/10
- 3s - loss: 9.5443e-04 - acc: 1.0000 - val_loss: 0.1070 - val_acc: 0.9773
Epoch 10/10
- 3s - loss: 3.9466e-04 - acc: 1.0000 - val_loss: 0.1126 - val_acc: 0.9769
```

```
#graphs
vhistory = [h1,h2]
vname = [model1.name, model2.name]

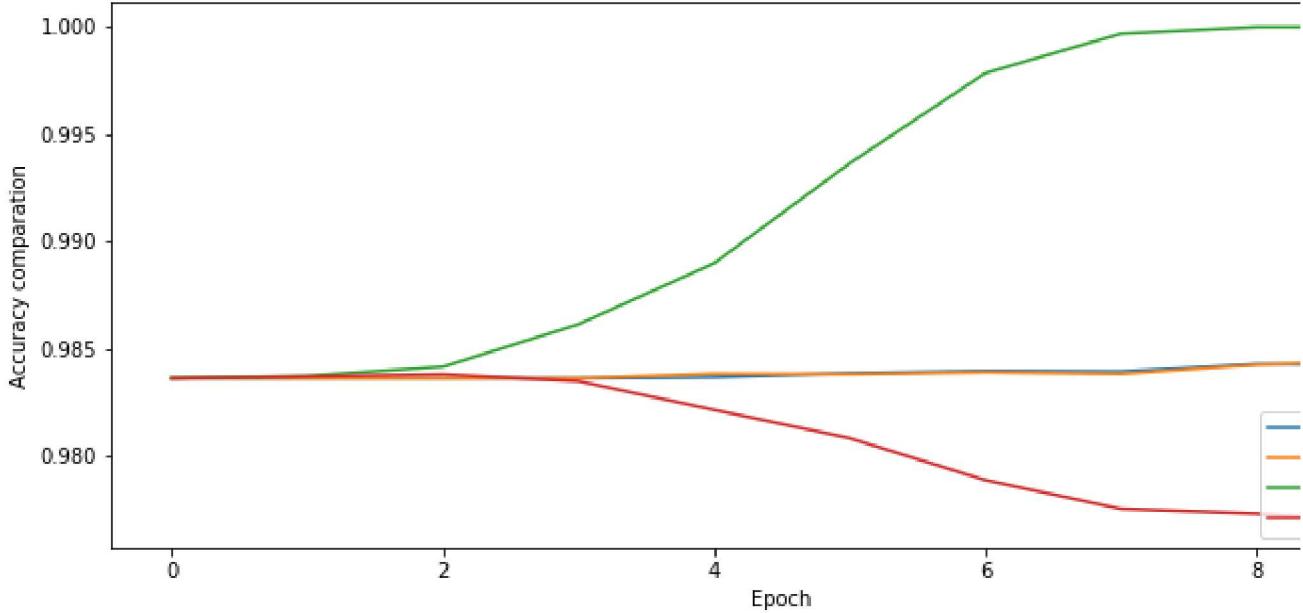
plot_compare_alb_loss_acc(vhistory, vname)
```

⟳

Loss comparation



Accuracy comparation



```
model1.predict(xtest).shape
```

⟳ (2000, 61)

Para ver cómo evoluciona nuestro modelo del lenguaje, vamos a generar texto según va entrenando, utilizando el modelo en su estado actual, genere texto, con la idea de ver cómo se va generando. En el código de abajo podemos ver una función auxiliar para obtener valores de una distribución y muestrear el siguiente carácter a utilizar según las probabilidades de la salida de softmax (en vez

máxima probabilidad, obtenemos un valor aleatorio según la distribución de probabilidad dada por probs. Los resultados serán más diversos, pero seguirán teniendo "sentido" ya que el modelo tenderá a seleccionar valores que tienen una probabilidad más alta.

```
def sample(probs, temperature=1.0):
    """Nos da el índice del elemento a elegir según la distribución
    de probabilidad dada por probs.

    Args:
        probs es la salida dada por una capa softmax:
        probs = model.predict(x_to_predict)[0]

        temperature es un parámetro que nos permite obtener mayor
        "diversidad" a la hora de obtener resultados.

        temperature = 1 nos da la distribución normal de softmax
        0 < temperature < 1 hace que el sampling sea más conservador,
        de modo que sampleamos cosas de las que estamos más seguros
        temperature > 1 hace que los samplings sean más atrevidos,
        eligiendo en más ocasiones clases con baja probabilidad.
        Con esto, tenemos mayor diversidad pero se cometen más
        errores.

    """
    # Cast a float64 por motivos numéricos
    probs = np.asarray(probs).astype('float64')

    # Hacemos logaritmo de probabilidades y aplicamos reducción
    # por temperatura.
    probs = np.log(probs) / temperature

    # Volvemos a aplicar exponencial y normalizamos de nuevo
    exp_probs = np.exp(probs)
    probs = exp_probs / np.sum(exp_probs)

    # Hacemos el sampling dadas las nuevas probabilidades
    # de salida (ver doc. de np.random.multinomial)
    samples = np.random.multinomial(1, probs, 1)
    return np.argmax(samples)
```

Utilizando la función anterior y el modelo entrenado, vamos a añadir un callback a nuestro modelo para generar textos con distintas temperaturas al acabar cada epoch.

Para ello, abajo tenéis disponible el callback `on_epoch_end`. Esta función elige una secuencia de temperaturas y genera textos de longitud `GENERATED_TEXT_LENGTH` según las temperaturas en la función `generate_text`.

Completa la función `generate_text` de modo que utilicemos el modelo y la función `sample` para generar textos.

NOTA: Cuando hagas `model.predict`, es aconsejable usar `verbose=0` como argumento para evitar que imprima información innecesaria.

```
TEMPERATURES_TO_TRY = [0.2, 0.5, 1.0, 1.2]
GENERATED_TEXT_LENGTH = 200
```

```
model = modelt
```

```
def generate_text(seed_text, model, length, temperature=1):
    """Genera una secuencia de texto a partir de seed_text utilizando model.

    La secuencia tiene longitud length y el sampling se hace con la temperature
    definida.
    """

    # Aquí guardaremos nuestro texto generado, que incluirá el
    # texto origen
    generated = seed_text

    # Utilizar el modelo en un bucle de manera que generaremos
    # carácter a carácter. Habrá que construir los valores de
    # X_pred de manera similar a como hemos hecho arriba, salvo que
    # aquí sólo se necesita una oración
    # Nótese que el x que utilicemos tiene que irse actualizando con
    # los caracteres que se van generando. La secuencia de entrada al
    # modelo tiene que ser una secuencia de tamaño SEQ_LENGTH que
    # incluya el último carácter predicho.
```

TU CÓDIGO AQUÍ

```
for j in range(0,length):
    # construimos la matriz
    X_pred = np.zeros((1, SEQ_LENGTH, NUM_CHARS))

    # lista de secuencias
    mseq = list(seed_text)

    # asignamos la entrada a X
    for k in range(0,SEQ_LENGTH):
        X_pred[0, k, dirdict[mseq[k]]] = 1

    # predecimos
    probs = model.predict(X_pred)[0]
    char_gen = sample(probs, temperature=temperature)

    generated = generated+invdict[char_gen]
    seed_text = seed_text[1:]+invdict[char_gen]

### FIN DE TU CÓDIGO

return generated
```

```
def on_epoch_end(epoch, logs):
    print("\n\n\n")
```

```
# Primero, seleccionamos una secuencia al azar para empezar a predecir
# a partir de ella
start_pos = random.randint(0, len(text) - SEQ_LENGTH - 1)
seed_text = text[start_pos : start_pos + SEQ_LENGTH]
```

```
seed_text = text[scalar_pos: scalar_pos + SEQ_LENGTH]
for temperature in TEMPERATURES_TO_TRY:
    print("-----> Epoch: {} - Generando texto con temperatura {}".format(epoch + 1, temperature))

    generated_text = generate_text(seed_text, model, GENERATED_TEXT_LENGTH, temperature)
    print("Seed: {}".format(seed_text))
    print("Tx g: {}".format(generated_text))
    print()

generation_callback = LambdaCallback(on_epoch_end=on_epoch_end)
```

Entrena ahora tu modelo. No te olvides de añadir `generation_callback` a la lista de callbacks utilizadas. Las clasificaciones no son tan críticas aquí (no nos importa tanto acertar el carácter exacto, sino obtenerlo). No es necesario monitorizar la accuracy ni usar validation data, si bien puedes añadirlos para así.

```
## TU CÓDIGO AQUÍ
#MAX_SEQUENCES = 500000

# Separmos para test y train
N = 40000
X = X[0:N,:,:]
y = y[0:N,:]

# Dividimos train en train/val
from sklearn.model_selection import train_test_split
perc = 0.4
xtrain, xval, ytrain, yval = train_test_split(X, y, test_size=perc)

model.fit(xtrain,
           ytrain,
           epochs = 10,
           verbose= 2,
           callbacks=[generation_callback],
           validation_data = [xval,yval])
```



Tx g: de sus padres, que medianamento caballora y esto me domer, y ser que su mendo y meros, o se sucho y su mara, y que la grigo esta de la himera de lla gantenta

-----> Epoch: 9 - Generando texto con temperature 1.0

Seed: de sus padres, que medianamen

Tx g: de sus padres, que medianamencusencaco lueste de jursos porgo que manda que cuaba; y poros, moyos,

uya.

cumo despar derla.

con quijo

camo de cozo que derco espente, coma de los despa calteso y fres dan manarna de le guasir

-----> Epoch: 9 - Generando texto con temperature 1.2

Seed: de sus padres, que medianamen

Tx g: de sus padres, que medianamendo: roñocdon el breco vestlos -dencembien omo hus mabía; y crosqueleme

le

gurdul a-se bles puegado es mizo este ser huco, y enquijo -dasí tíaicomple, un tuerían

vue esde su ratá

Epoch 10/10

- 44s - loss: 0.0447 - acc: 0.9856 - val_loss: 0.0446 - val_acc: 0.9856

-----> Epoch: 10 - Generando texto con temperature 0.2

Seed: manos, y ponía mano a la espad

Tx g: manos, y ponía mano a la espada de su señor a la manción de la viciando de la m

-----> Epoch: 10 - Generando texto con temperature 0.5

Seed: manos, y ponía mano a la espad

Tx g: manos, y ponía mano a la espada de la meñor vero señor el cual viera a cancial]

-----> Epoch: 10 - Generando texto con temperature 1.0

Seed: manos, y ponía mano a la espad

Tx g: manos, y ponía mano a la espada son quijote valos por mevir señor elguica los r en esto a que el sodalles de mazaras, asíse por ancira de su pomo que la d

-----> Epoch: 10 - Generando texto con temperature 1.2

Seed: manos, y ponía mano a la espad

Tx g: manos, y ponía mano a la espada- la cuino mi quesjo; y son señor viisen armanc

-mon lo

vinge de

porrea

gostalde que escenta. en varque buotó llevamo

<keras.callbacks.History at 0x7fd614937828>

