RNN

Joel Isaias Solano Ocampo | A01639289

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
import os
```

Descarga de datos

In [2]:
```
path_to_fileDL = tf.keras.utils.get_file('pg90.txt', 'https://www.gutenberg.org/cache/

text = open(path_to_fileDL, 'rb').read().decode(encoding='utf-8')
print('Longitud del texto:    {} caracteres'.format(len(text)))

vocab = sorted(set(text))
print ('El texto esta compuesto de estos {} caracteres'.format(len(vocab)))
print (vocab)
```

```
Downloading data from https://www.gutenberg.org/cache/epub/90/pg90.txt
556635/556635 [==============================] - 1s 2us/step
Longitud del texto:    550671 caracteres
El texto esta compuesto de estos 91 caracteres
['\n', '\r', ' ', '!', '#', '$', '%', '(', ')', '*', ',', '-', '.', '/', '0', '1',
'2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F',
'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', '[', ']', '_', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'é', '—',
'‘', '’', '“', '”', '•', '™', '\ufeff']
```

Tablas de traduccion o Inversa de vocabulario

In [3]:
```
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)
```

In [4]:
```
for char,_ in zip(char2idx, range(len(vocab))):
  print(' {:4s}: {:3d},'.format(repr(char), char2idx[char]))
```

```
'\n':   0,
'\r':   1,
' ' :   2,
'!' :   3,
'#' :   4,
'$' :   5,
'%' :   6,
'(' :   7,
')' :   8,
'*' :   9,
',' :  10,
'-' :  11,
'.' :  12,
'/' :  13,
'0' :  14,
'1' :  15,
'2' :  16,
'3' :  17,
'4' :  18,
'5' :  19,
'6' :  20,
'7' :  21,
'8' :  22,
'9' :  23,
':' :  24,
';' :  25,
'?' :  26,
'A' :  27,
'B' :  28,
'C' :  29,
'D' :  30,
'E' :  31,
'F' :  32,
'G' :  33,
'H' :  34,
'I' :  35,
'J' :  36,
'K' :  37,
'L' :  38,
'M' :  39,
'N' :  40,
'O' :  41,
'P' :  42,
'Q' :  43,
'R' :  44,
'S' :  45,
'T' :  46,
'U' :  47,
'V' :  48,
'W' :  49,
'X' :  50,
'Y' :  51,
'Z' :  52,
'[' :  53,
']' :  54,
'_' :  55,
'a' :  56,
'b' :  57,
'c' :  58,
'd' :  59,
```

```
 'e' :  60,
 'f' :  61,
 'g' :  62,
 'h' :  63,
 'i' :  64,
 'j' :  65,
 'k' :  66,
 'l' :  67,
 'm' :  68,
 'n' :  69,
 'o' :  70,
 'p' :  71,
 'q' :  72,
 'r' :  73,
 's' :  74,
 't' :  75,
 'u' :  76,
 'v' :  77,
 'w' :  78,
 'x' :  79,
 'y' :  80,
 'z' :  81,
 'é' :  82,
 '—' :  83,
 '‘' :  84,
 '’' :  85,
 '“' :  86,
 '”' :  87,
 '•' :  88,
 '™' :  89,
 '\ufeff':  90,
```

convertir texto a enteros

In [5]:
```
text_as_int = np.array([char2idx[c] for c in text])
```

In [6]:
```
#Mostramos algunos caracteres
print('text: {}'.format(repr(text[:50])))
print('{}'.format(repr(text_as_int[:50])))
```

```
text: '\ufeffThe Project Gutenberg eBook of The Son of Tarzan\r'
array([90, 46, 63, 60,  2, 42, 73, 70, 65, 60, 58, 75,  2, 33, 76, 75, 60,
       69, 57, 60, 73, 62,  2, 60, 28, 70, 70, 66,  2, 70, 61,  2, 46, 63,
       60,  2, 45, 70, 69,  2, 70, 61,  2, 46, 56, 73, 81, 56, 69,  1])
```

PREPARAR DATOS

In [7]:
```
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

seq_length = 100

sequences = char_dataset.batch(seq_length+1, drop_remainder=True)
```

In [8]:
```
#comprobar datos
for item in sequences.take(10):
  print(repr(''.join(idx2char[item.numpy()])))
```

```
'\ufeffThe Project Gutenberg eBook of The Son of Tarzan\r\n    \r\nThis ebook is for
the use of anyone anywhere'
' in the United States and\r\nmost other parts of the world at no cost and with almos
t no restrictions\r\n'
'whatsoever. You may copy it, give it away or re-use it under the terms\r\nof the Pro
ject Gutenberg Lice'
'nse included with this ebook or online\r\nat www.gutenberg.org. If you are not locat
ed in the United St'
'ates,\r\nyou will have to check the laws of the country where you are located\r\nbef
ore using this eBook.'
'\r\n\r\nTitle: The Son of Tarzan\r\n\r\n\r\nAuthor: Edgar Rice Burroughs\r\n\r\nRele
ase date: November 1, 1993 [eBo'
'ok #90]\r\n                Most recently updated: June 23, 2022\r\n\r\nLanguage: Eng
lish\r\n\r\n\r\n\r\n*** START O'
'F THE PROJECT GUTENBERG EBOOK THE SON OF TARZAN ***\r\n[Illustration]\r\n\r\n\r\n\r
\n\r\nThe Son Of Tarzan\r\n\r\nby '
'Edgar Rice Burroughs\r\n\r\n\r\n\r\n\r\nTo Hulbert Burroughs\r\n\r\n\r\n\r\nCont
ents\r\n\r\n I.\r\n II.\r\n III.\r\n IV.\r\n V.\r\n'
' VI.\r\n VII.\r\n VIII.\r\n IX.\r\n X.\r\n XI.\r\n XII.\r\n XIII.\r\n XIV.\r\n XV.\r
\n XVI.\r\n XVII.\r\n XVIII.\r\n XIX.\r\n XX.'
```

In [9]:
```
#Preparar datos de entrenamiento  (Entrada 0 a 99 )  (Salida 1 a 100)
def split_input_target(chunk):
  input_text = chunk[:-1]
  target_text = chunk[1:]
  return input_text, target_text

dataset = sequences.map(split_input_target)
```

In [10]:
```
#Visualizamos
for input_example, target_example in dataset.take(1):
  print ('Input data: ', repr(''.join(idx2char[input_example.numpy()])))
  print ('Target data: ', repr(''.join(idx2char[target_example.numpy()])))
```

```
Input data:  '\ufeffThe Project Gutenberg eBook of The Son of Tarzan\r\n    \r\nThis
ebook is for the use of anyone anywher'
Target data:  'The Project Gutenberg eBook of The Son of Tarzan\r\n    \r\nThis ebook
is for the use of anyone anywhere'
```

In [11]:
```
#imprimir dataset
print (dataset)
```

```
<_MapDataset element_spec=(TensorSpec(shape=(100,), dtype=tf.int64, name=None), Tenso
rSpec(shape=(100,), dtype=tf.int64, name=None))>
```

In [12]:
```
#agrupar en batches
BATCH_SIZE = 64
BUFFER_SIZE = 10000

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
print(dataset)
```

```
<_BatchDataset element_spec=(TensorSpec(shape=(64, 100), dtype=tf.int64, name=None),
TensorSpec(shape=(64, 100), dtype=tf.int64, name=None))>
```

Construir modelo RNN

In [13]:
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
  model = tf.keras.Sequential([
      tf.keras.layers.Embedding(vocab_size, embedding_dim,
```

```
                                batch_input_shape=[batch_size,None]),

      tf.keras.layers.LSTM(rnn_units,
                           return_sequences=True,
                           stateful = True,
                           recurrent_initializer='glorot_uniform'),

      tf.keras.layers.Dense(vocab_size)
  ])
  return model

vocab_size = len(vocab)
embedding_dim= 256
rnn_units = 1024

model = build_model(
    vocab_size = vocab_size,
    embedding_dim=embedding_dim,
    rnn_units=rnn_units,
    batch_size = BATCH_SIZE
)
```

In [14]:
```
#Visualizar estructura
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (64, None, 256)           23296

 lstm (LSTM)                 (64, None, 1024)          5246976

 dense (Dense)               (64, None, 91)            93275

=================================================================
Total params: 5363547 (20.46 MB)
Trainable params: 5363547 (20.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [15]:
```
# Forma de input
for input_example_batch, target_example_batch in dataset.take(1):
  print("Input: ", input_example_batch.shape, "# (batch_size, lenght)")
  print("Target: ", target_example_batch.shape, "# (batch_size, sequence_length)")
```

```
Input:  (64, 100) # (batch_size, lenght)
Target:  (64, 100) # (batch_size, sequence_length)
```

In [16]:
```
#Forma de salida
for input_example_batch, target_example_batch in dataset.take(1):
  example_batch_predictions = model(input_example_batch)
  print("Prediction: ", example_batch_predictions.shape, "# (batch_size, sequence_leng
```

```
Prediction:  (64, 100, 91) # (batch_size, sequence_length, vocab_size)
```

In [17]:
```
#Mostar que el resultado es una distribucion, no un argmax

sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
```

```
sampled_indices_characters = tf.squeeze(sampled_indices,axis=-1).numpy()
print(sampled_indices_characters)
```

```
[90 68 32 37 28 75 85  4 54 20 53 61 36 72 74 18 83 63 87 71 57 26 70  0
 44 43 60 77 86 52 76 77  4 69 48 90 50 32 22 13 48 40 77 51 83 90 40 13
  4 56 21 64 14  0 45 28 24 75 11 90 51 47 67 47  4 76 71 65 56 44 67 53
 85 39 66  0 18 60 10 62 43 47 70 37  3 47 71 20 43 48 29 85 31 82 62 45
 67 71 24  2]
```

ENTRENAMIENTO

In [18]:
```
def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=T

model.compile(optimizer='adam', loss=loss)
```

In [19]:
```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_(epoch)")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True
)
```

In [20]:
```
EPOCHS = 50

history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/50
85/85 [==============================] - 10s 79ms/step - loss: 2.8155
Epoch 2/50
85/85 [==============================] - 8s 72ms/step - loss: 2.1405
Epoch 3/50
85/85 [==============================] - 7s 73ms/step - loss: 1.8638
Epoch 4/50
85/85 [==============================] - 7s 69ms/step - loss: 1.6707
Epoch 5/50
85/85 [==============================] - 7s 70ms/step - loss: 1.5351
Epoch 6/50
85/85 [==============================] - 7s 71ms/step - loss: 1.4333
Epoch 7/50
85/85 [==============================] - 7s 71ms/step - loss: 1.3593
Epoch 8/50
85/85 [==============================] - 7s 73ms/step - loss: 1.3041
Epoch 9/50
85/85 [==============================] - 7s 73ms/step - loss: 1.2564
Epoch 10/50
85/85 [==============================] - 7s 74ms/step - loss: 1.2142
Epoch 11/50
85/85 [==============================] - 8s 75ms/step - loss: 1.1761
Epoch 12/50
85/85 [==============================] - 7s 76ms/step - loss: 1.1384
Epoch 13/50
85/85 [==============================] - 7s 75ms/step - loss: 1.1024
Epoch 14/50
85/85 [==============================] - 7s 78ms/step - loss: 1.0676
Epoch 15/50
85/85 [==============================] - 7s 78ms/step - loss: 1.0312
Epoch 16/50
85/85 [==============================] - 8s 79ms/step - loss: 0.9922
Epoch 17/50
85/85 [==============================] - 8s 79ms/step - loss: 0.9517
Epoch 18/50
85/85 [==============================] - 8s 79ms/step - loss: 0.9129
Epoch 19/50
85/85 [==============================] - 7s 77ms/step - loss: 0.8719
Epoch 20/50
85/85 [==============================] - 7s 76ms/step - loss: 0.8274
Epoch 21/50
85/85 [==============================] - 8s 76ms/step - loss: 0.7854
Epoch 22/50
85/85 [==============================] - 8s 79ms/step - loss: 0.7429
Epoch 23/50
85/85 [==============================] - 8s 79ms/step - loss: 0.7019
Epoch 24/50
85/85 [==============================] - 8s 79ms/step - loss: 0.6619
Epoch 25/50
85/85 [==============================] - 7s 78ms/step - loss: 0.6220
Epoch 26/50
85/85 [==============================] - 7s 77ms/step - loss: 0.5887
Epoch 27/50
85/85 [==============================] - 8s 75ms/step - loss: 0.5532
Epoch 28/50
85/85 [==============================] - 7s 76ms/step - loss: 0.5232
Epoch 29/50
85/85 [==============================] - 7s 77ms/step - loss: 0.4961
Epoch 30/50
85/85 [==============================] - 7s 77ms/step - loss: 0.4719
```

```
Epoch 31/50
85/85 [==============================] - 8s 77ms/step - loss: 0.4473
Epoch 32/50
85/85 [==============================] - 7s 78ms/step - loss: 0.4279
Epoch 33/50
85/85 [==============================] - 8s 79ms/step - loss: 0.4102
Epoch 34/50
85/85 [==============================] - 8s 78ms/step - loss: 0.3935
Epoch 35/50
85/85 [==============================] - 8s 79ms/step - loss: 0.3802
Epoch 36/50
85/85 [==============================] - 7s 76ms/step - loss: 0.3666
Epoch 37/50
85/85 [==============================] - 8s 76ms/step - loss: 0.3559
Epoch 38/50
85/85 [==============================] - 7s 76ms/step - loss: 0.3483
Epoch 39/50
85/85 [==============================] - 7s 77ms/step - loss: 0.3382
Epoch 40/50
85/85 [==============================] - 7s 78ms/step - loss: 0.3303
Epoch 41/50
85/85 [==============================] - 8s 78ms/step - loss: 0.3252
Epoch 42/50
85/85 [==============================] - 8s 79ms/step - loss: 0.3173
Epoch 43/50
85/85 [==============================] - 8s 79ms/step - loss: 0.3137
Epoch 44/50
85/85 [==============================] - 8s 78ms/step - loss: 0.3058
Epoch 45/50
85/85 [==============================] - 8s 77ms/step - loss: 0.3032
Epoch 46/50
85/85 [==============================] - 7s 77ms/step - loss: 0.2973
Epoch 47/50
85/85 [==============================] - 7s 77ms/step - loss: 0.2954
Epoch 48/50
85/85 [==============================] - 7s 77ms/step - loss: 0.2896
Epoch 49/50
85/85 [==============================] - 8s 76ms/step - loss: 0.2870
Epoch 50/50
85/85 [==============================] - 7s 77ms/step - loss: 0.2822
```

Generacion de texto

In [21]:
```
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)

model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))

model.build(tf.TensorShape([1,None]))
```

In [22]:
```
def generate_text(model, start_string, temp):
  num_generate = 500
  input_eval = [char2idx[s] for s in start_string]

  input_eval = tf.expand_dims(input_eval, 0)
  text_generated = []

  temperature = temp

  model.reset_states()
```

```
  for i in range(num_generate):
    predictions = model(input_eval)

    predictions = tf.squeeze(predictions,0)

    predictions = predictions/temperature
    predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()

    input_eval = tf.expand_dims([predicted_id],0)

    text_generated.append(idx2char[predicted_id])

  return(start_string + ''.join(text_generated))
```

In [26]:
```
print(generate_text(model, start_string=u"tarzan", temp=0.5))
```

```
tarzan king, for the boy wishes in which the
photograph of a little girl upon the bed country where The Sheik held him back, and
form the right with the handsome face, and there, for an angry
relations to the north the Big Bwana and his black warriors clung tenaciously
to the trail of the lesson attempted to analyse the relationship than that of them.

Korak landed with the exclusion of the jungle. He would not have thought of himself a
t all; but the egotism of the house men—Mr. Moore recognized
```

In [27]:
```
print(generate_text(model, start_string=u"tarzan", temp=1))
```

```
tarzan’s wildsome them.

Korak saw the man before him, turned back toward the
village. Here was the boat thus day the beast’s hangings that had
defeared him. Uniffed battle with the agility of a Cannot lorat
Of London, where, he shouted to
his captive that the blacks followed him, and
this listening flight for the
great Akut and at a dorge

and and his little eyes because of his soul of the old wealok took him more
of the clearing and developed in the direction of the thing that had been in
```

In [28]:
```
print(generate_text(model, start_string=u"son", temp=1))
```

```
son on which she had tet her indicatory of the Swede.

As the Hon. Morison laughed uneasily. He didn’t care to appear at a
disadvantage before this giant and the
yellow-greater of filthy dampered trump within his
brd Baynes, and but the two
halted fig an ungramment. If you do not agree to abide by all
the adert for any mor the strangers.
How he loved? She drew back in the enclosure a
mighty north almost the coast of it; but I may so Lord Greystoke, please, Molerous
my eater at home.”

“
```

In [29]:
```
print(generate_text(model, start_string=u"son", temp=2))
```

```
son.”

Then he dragged him killingnesse-might little trademabls
upon Jenssen. He through the timl? Lost nail yow; but that he epcive
if she was!”

It was liOked; to him k oSAkumeabl, phy dailty comp warm.

AT TH[AR, the Mangani, who macked them
Forgative Panlim hesitauld, as “but ah, Ge?,2” are camp forball
hunting heret
jungle mage—it,” he commanded. “Welitu puny or questions of the lad, and, MhoD
All rided skrnow more
ought them at knie, but AfCtibut is husce, troups, give him
That
```

In [30]:
```
print(generate_text(model, start_string=u"father", temp=2))
```

```
father,” angrmels

My his own, and.

“NDo,”) rash at no sign of dather she e.’Qwey’s faces of geather.”

He gavela and cun help? Ing nazed and lowful Geeka whinded iFw/crossing
in tivet, the For his Eku?”

“NDa, Vo0 M8 zandly. But, some
friend he e Proj O™n enemieff u™w
cross/be, bo,” ald the Hon. Morison noticed n knew escape wit
but the ape? As held
Har
Secr them—he tostipuls, dotelve it, after goasu, and princitatofF” and run to follow
the is creeis
vaduated d attacked them y
slighte
```

In [31]:
```
print(generate_text(model, start_string=u"son", temp=0.5))
```

```
son
one of the music hall and see Ajax, with a length of ancient slave chain fastened at
one end to an iron
collar padlocked about her nose at the addres
toward the North Baynes leaped their flying
mounts through the breach in the palisade and were gone up the
well-work, shortly as though it and what he had the chosurath and annively shribl, of
cient and the two ponies. He
had been with Malbihn for a year. The baboons continued to take her with
him by force rather than stolen place a few fee
```