INSTITUTO TECNOLÓGICO DE IZTAPALAPA

Ingeniería en Sistemas Computacionales

Generación de texto con una RNN

Proyecto Final de I.A.

De la asignatura: Inteligencia Artificial.

Profesor: Abiel Tomas Parra Hernández

Ángel Arellano Cabrera
Ángel de Jesús Alonso Ayala
Enrique Lopez Ruano
ISC-8AV

Índice.

Generación de texto con una RNN (REDES NEURONALES RECURRENTES – RECURRENT NEURAL NETWORKS).

Introducción.

Este proyecto muestra cómo generar texto usando un RNN basado en caracteres. Trabajaremos con un conjunto de datos de los escritos de Shakespeare de La irrazonable efectividad de las redes neuronales recurrentes de Andrej Karpathy. Dada una secuencia de caracteres a partir de estos datos ("Shakespear"), entrenamos un modelo para predecir el siguiente carácter en la secuencia ("e"). Se pueden generar secuencias de texto más largas llamando al modelo repetidamente.

Cabe señalar que, si bien algunas de las oraciones son gramaticales, la mayoría no tiene sentido. El modelo no ha aprendido el significado de las palabras, pero considere:

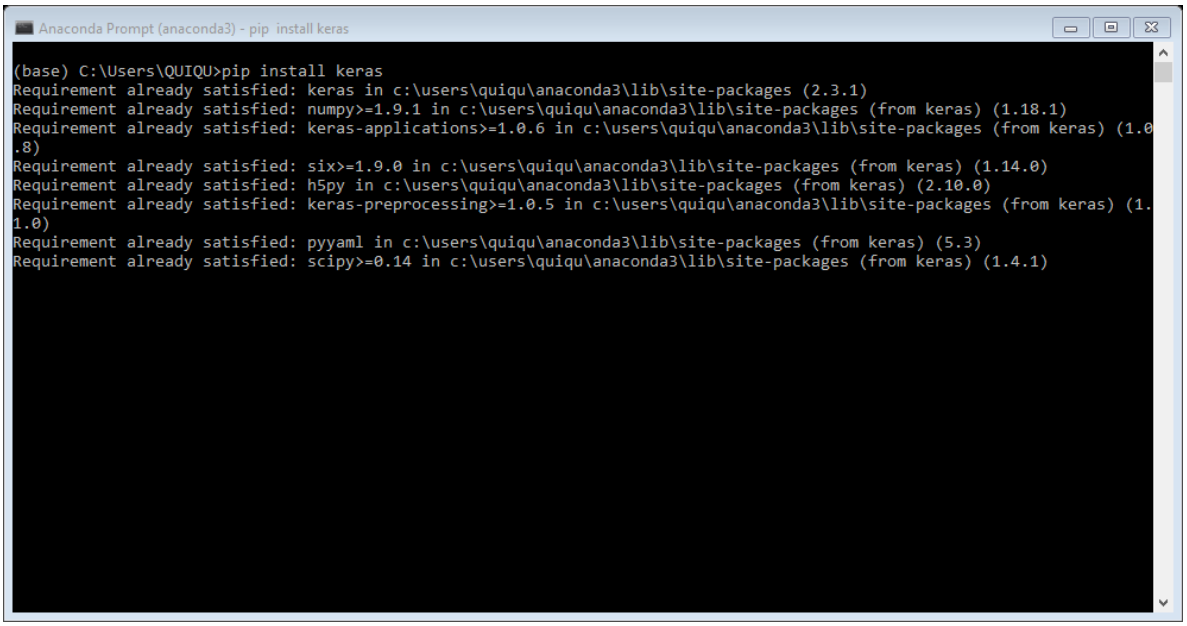
- El modelo está basado en personajes. Cuando comenzó el entrenamiento, la modelo no sabía cómo deletrear una palabra en inglés, o esas palabras eran incluso una unidad de texto.
- La estructura de la salida se asemeja a una obra de teatro: los bloques de texto generalmente comienzan con el nombre del hablante, en mayúsculas, similar al conjunto de datos.
- Como se demuestra a continuación, el modelo está entrenado en pequeños lotes de texto (100 caracteres cada uno) y aún puede generar una secuencia de texto más larga con una estructura coherente.

La elaboración de este proyecto fue en Python con las librerías de TensorFlow, Keras e instalación de la plataforma anaconda.

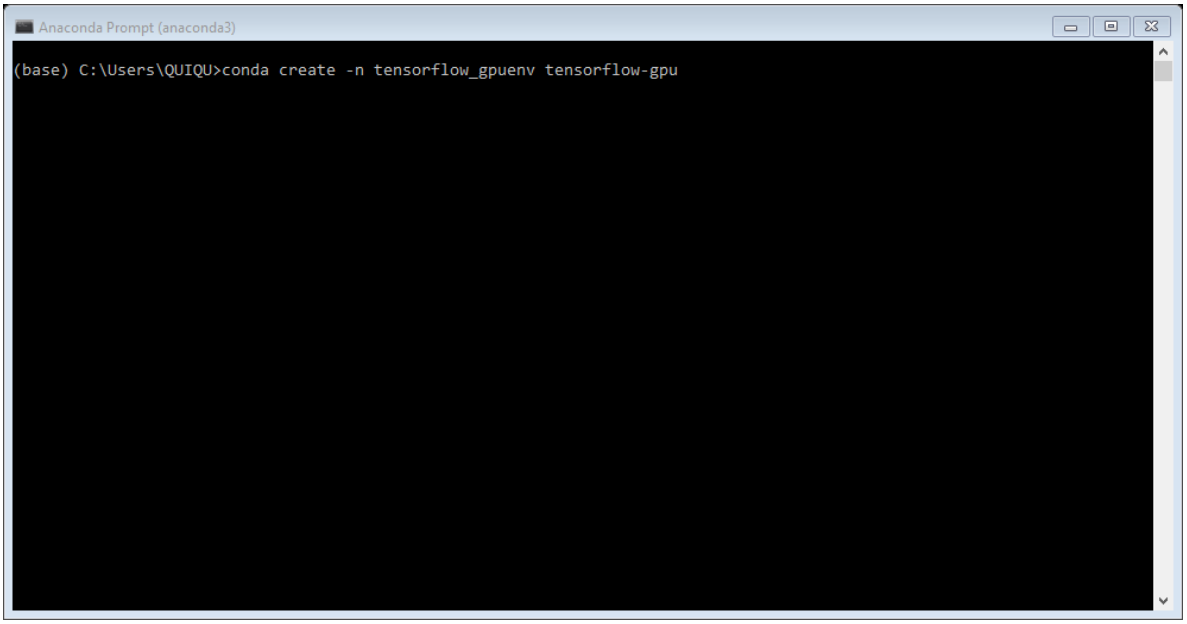
Desarrollo.

Como primer paso para el desarrollo del proyecto es instalar la plataforma Anaconda, ya que en esta se escribirá el modelado de las estructuras del proyecto e instalación de librerías que servirán para que el resultado sea correcto.

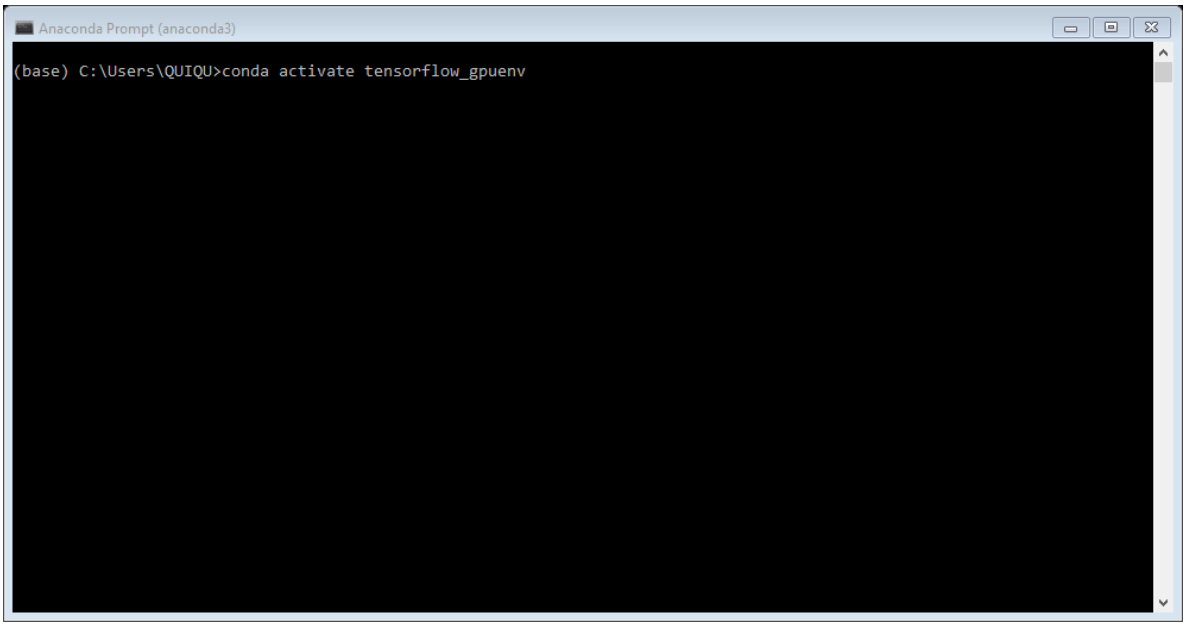
Posteriormente se instala por medio de la consola de anaconda la librería keras:



Luego instalamos el TensorFlow para GPU desde consola de Anaconda:



Se activa TensorFlow en GPU.



Posteriormente abrimos la extensión de anaconda llamada Spyder para empezar a escribir nuestro código para el proyecto.

Pasos para la elaboración del modelo:

- Importar TensorFlow y otras bibliotecas:

```
import tensorflow as tf

import numpy as np
import os
import time
```

- Descargar el conjunto de datos de Shakespeare

```
path_to_file = tf.keras.utils.get_file('shakespeare.txt',
'https://storage.googleapis.com/download.tensorflow.org/data/shakespea
re.txt')
```

- Lee los datos

Primero, mira en el texto:

```
# Read, then decode for py2 compat.// Lea, luego decodifique para py2
compat.
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
# length of text is the number of characters in it//la longitud del
texto es la cantidad de caracteres que contiene
print ('Length of text: {} characters'.format(len(text)))


# Take a look at the first 250 characters in text//Echa un vistazo a los
primeros 250 caracteres del texto.

print(text[:250])


# The unique characters in the file// Los caracteres únicos en el archivo
vocab = sorted(set(text))
print ('{} unique characters'.format(len(vocab)))
```

- Procesar el texto

Vectoriza el texto:

Antes de entrenar, necesitamos asignar cadenas a una representación numérica. Cree dos tablas de búsqueda: una asignación de caracteres a números y otra para números a caracteres.

```
# Creating a mapping from unique characters to índices// Crear un mapeo
de caracteres únicos a índices

char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

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

Ahora tenemos una representación entera para cada personaje. Observe que asignamos el carácter como índices de 0 a len (único).

```
print('{')
for char,_ in zip(char2idx, range(20)):
    print('  {:4s}: {:3d},'.format(repr(char), char2idx[char]))
print('  ...\n}')
```

```
{
  '\n':   0,
  ' ' :   1,
  '!' :   2,
```

```
  '$' :   3,
  '&' :   4,
  "'" :   5,
  ',' :   6,
  '-' :   7,
  '.' :   8,
  '3' :   9,
  ':' :  10,
  ';' :  11,
  '?' :  12,
  'A' :  13,
  'B' :  14,
  'C' :  15,
  'D' :  16,
  'E' :  17,
  'F' :  18,
  'G' :  19,
  ...
}
```

```
# Show how the first 13 characters from the text are mapped to
integers// Muestra cómo los primeros 13 caracteres del texto se
asignan a enteros
print ('{} ---- characters mapped to int ---- >
{}'.format(repr(text[:13]), text_as_int[:13]))
```

```
'First Citizen' ---- characters mapped to int ---- > [18 47 56 57 58
1 15 47 58 47 64 43 52]
```

- La tarea de predicción.

Dado un personaje, o una secuencia de caracteres, ¿cuál es el próximo personaje más probable? Esta es la tarea que estamos entrenando para que realice el modelo. La entrada al modelo será una secuencia de caracteres, y entrenamos al modelo para predecir la salida: el siguiente carácter en cada paso de tiempo.

Dado que los RNN mantienen un estado interno que depende de los elementos vistos anteriormente, dados todos los caracteres calculados hasta este momento, ¿cuál es el siguiente carácter?

- Crea ejemplos y objetivos de entrenamiento.

Luego divida el texto en secuencias de ejemplo. Cada secuencia de entrada contendrá seq_length caracteres del texto.

Para cada secuencia de entrada, los objetivos correspondientes contienen la misma longitud de texto, excepto que se desplaza un carácter a la derecha.

Así que divida el texto en trozos de seq_length + 1. Por ejemplo, digamos que seq_length es 4 y nuestro texto es "Hola". La secuencia de entrada sería "Infierno" y la secuencia de destino "ello".

Para hacer esto, primero use la función tf.data.Dataset.from_tensor_slices para convertir el vector de texto en una secuencia de índices de caracteres.

```
# The maximum length sentence we want for a single input in
characters// La oración de longitud máxima que queremos para una sola
entrada en caracteres
seq_length = 100
examples_per_epoch = len(text)//(seq_length+1)

# Create training examples / targets // Crear ejemplos / objetivos de
entrenamiento
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

for i in char_dataset.take(5):
  print(idx2char[i.numpy()])
```

```
F
i
r
s
t
```

El método (batch) por lotes nos permite convertir fácilmente estos caracteres individuales en secuencias del tamaño deseado.

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

for item in sequences.take(5):
  print(repr(''.join(idx2char[item.numpy()])))
```

```
'First Citizen:\nBefore we proceed any further, hear me
speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
'are all resolved rather to die than to famish?\n\nAll:\nResolved.
resolved.\n\nFirst Citizen:\nFirst, you k'
"now Caius Marcius is chief enemy to the people.\n\nAll:\nWe know't,
we know't.\n\nFirst Citizen:\nLet us ki"
"ll him, and we'll have corn at our own price.\nIs't a
verdict?\n\nAll:\nNo more talking on't; let it be d"
```

```
'one: away, away!\n\nSecond Citizen:\nOne word, good
citizens.\n\nFirst Citizen:\nWe are accounted poor citi'
```

Para cada secuencia, duplíquela y cámbiela para formar el texto de entrada y de destino utilizando el método de mapa para aplicar una función simple a cada lote:

```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)
```

Imprima los primeros ejemplos de entrada y valores objetivo:

```
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:  'First Citizen:\nBefore we proceed any further, hear me
speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou'
Target data: 'irst Citizen:\nBefore we proceed any further, hear me
speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
```

Cada índice de estos vectores se procesa como un paso de tiempo. Para la entrada en el paso de tiempo 0, el modelo recibe el índice para "F" e intenta predecir el índice para "i" como el siguiente carácter. En el siguiente paso de tiempo, hace lo mismo, pero el RNN considera el contexto del paso anterior además del carácter de entrada actual.

```
for i, (input_idx, target_idx) in enumerate(zip(input_example[:5],
target_example[:5])):
    print("Step {:4d}".format(i))
    print("  input: {} ({:s})".format(input_idx,
repr(idx2char[input_idx])))
    print("  expected output: {} ({:s})".format(target_idx,
repr(idx2char[target_idx])))
```

```
Step    0
  input: 18 ('F')
  expected output: 47 ('i')
Step    1
```

```
  input: 47 ('i')
  expected output: 56 ('r')
Step    2
  input: 56 ('r')
  expected output: 57 ('s')
Step    3
  input: 57 ('s')
  expected output: 58 ('t')
Step    4
  input: 58 ('t')
  expected output: 1 (' ')
```

- Crea lotes de entrenamiento

Utilizamos tf.data para dividir el texto en secuencias manejables. Pero antes de introducir estos datos en el modelo, necesitamos mezclar los datos y empaquetarlos en lotes.

```
# Batch size /Tamaño del lote/
BATCH_SIZE = 64

# Buffer size to shuffle the dataset //Tamaño del búfer para barajar
el conjunto de datos
# (TF data is designed to work with possibly infinite sequences,//
(Los datos TF están diseñados para funcionar con secuencias
posiblemente infinitas,
# so it doesn't attempt to shuffle the entire sequence in memory.
Instead,// para que no intente barajar toda la secuencia en la
memoria. En lugar,
# it maintains a buffer in which it shuffles elements).// mantiene un
búfer en el que baraja elementos).
BUFFER_SIZE = 10000

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

dataset

<BatchDataset shapes: ((64, 100), (64, 100)), types: (tf.int64,
tf.int64)>
```

- Construye el modelo

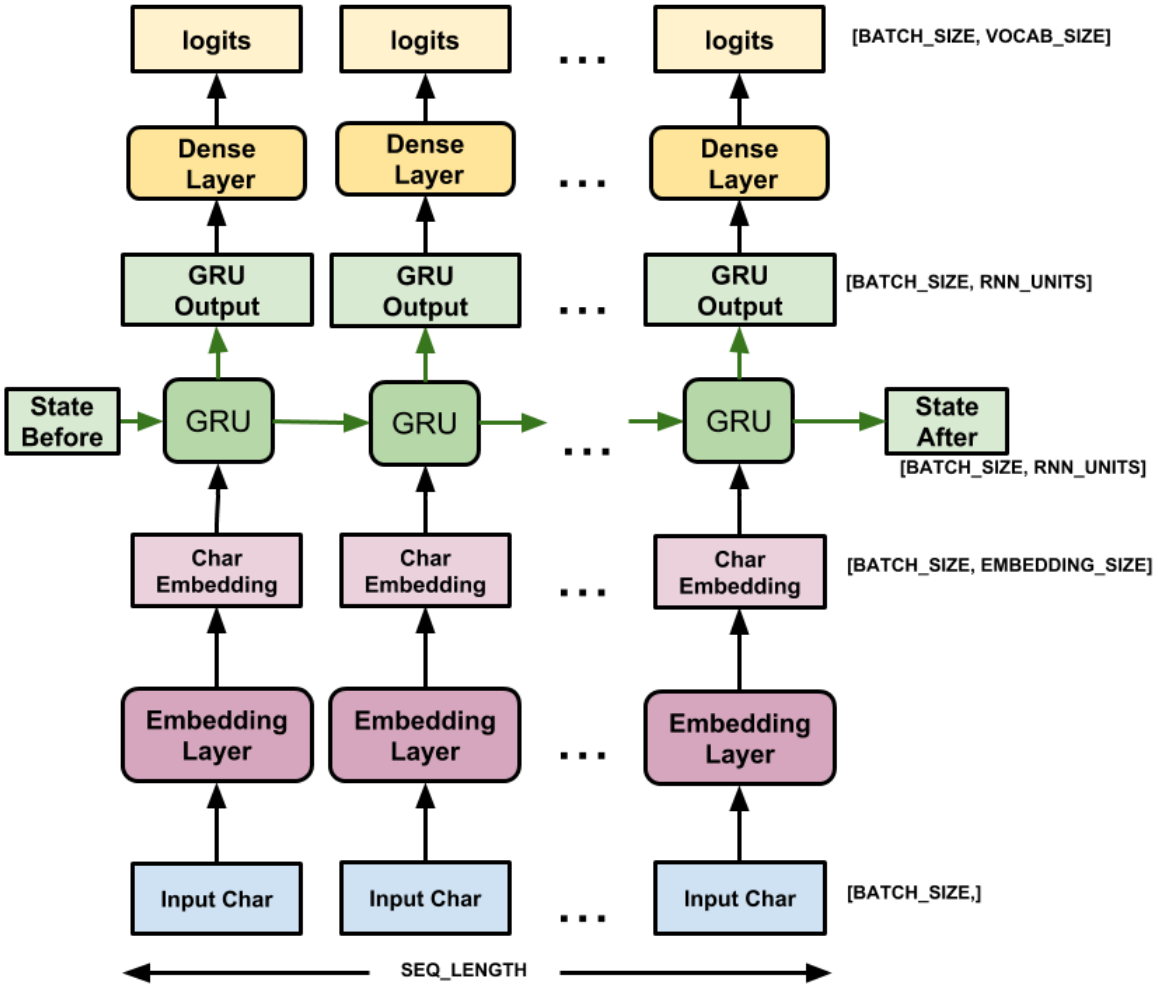
Use tf.keras.Sequential para definir el modelo. Para este sencillo ejemplo, se utilizan tres capas para definir nuestro modelo:

- tf.keras.layers.Embedding: la capa de entrada. Una tabla de búsqueda entrenable que asignará los números de cada carácter a un vector con dimensiones embedded_dim;
- tf.keras.layers.GRU: un tipo de RNN con unidades de tamaño = rnn_units (también puede usar una capa LSTM aquí).
- tf.keras.layers.Dense: la capa de salida, con salidas vocab_size.

```
# Length of the vocabulary in chars// Longitud del vocabulario en
caracteres.
vocab_size = len(vocab)

# The embedding dimensión // La dimensión de incrustación
embedding_dim = 256

# Number of RNN units // Cantidad de unidades RNN
rnn_units = 1024
```

```
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.GRU(rnn_units,
                        return_sequences=True,
                        stateful=True,
                        recurrent_initializer='glorot_uniform'),
    tf.keras.layers.Dense(vocab_size)
  ])
  return model

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

Para cada carácter, el modelo busca la incrustación, ejecuta el GRU una vez con la incrustación como entrada y aplica la capa densa para generar logits que predicen la probabilidad de registro del siguiente carácter:



Tenga en cuenta que elegimos el modelo secuencial de Keras aquí, ya que todas las capas del modelo solo tienen una sola entrada y producen una sola salida. En caso de que desee recuperar y reutilizar los estados de la capa RNN con estado, es posible que desee construir su modelo con la API funcional de Keras o la subclase de modelos. Por favor, consulte la guía Keras RNN para más detalles.

- Prueba el modelo

Ahora ejecute el modelo para ver que se comporta como se esperaba.

Primero verifique la forma de la salida:

```
for input_example_batch, target_example_batch in dataset.take(1):
  example_batch_predictions = model(input_example_batch)
  print(example_batch_predictions.shape, "# (batch_size,
sequence_length, vocab_size)")
```

```
(64, 100, 65) # (batch_size, sequence_length, vocab_size)
```

En el ejemplo anterior, la longitud de secuencia de la entrada es 100 pero el modelo se puede ejecutar en entradas de cualquier longitud:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (64, None, 256)           16640
_________________________________________________________________
gru (GRU)                    (64, None, 1024)          3938304
_________________________________________________________________
dense (Dense)                (64, None, 65)            66625
=================================================================
Total params: 4,021,569
Trainable params: 4,021,569
Non-trainable params: 0
_________________________________________________________________
```

Para obtener predicciones reales del modelo, necesitamos tomar muestras de la distribución de salida, para obtener índices de caracteres reales. Esta distribución está definida por los logits sobre el vocabulario de los personajes.

Nota: Es importante tomar muestras de esta distribución, ya que tomar la argmax de la distribución puede hacer que el modelo quede atascado en un bucle.

Pruébelo para el primer ejemplo en el lote:

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

Esto nos da, en cada paso de tiempo, una predicción del siguiente índice de caracteres:

```
sampled_indices
```

```
array([51, 23, 37,  5, 46, 55, 22,  1,  0,  3, 17, 38, 62, 25, 52, 57,
49,
       49, 53, 54, 41, 25, 63, 24, 16, 48, 28, 61, 54, 28, 35,  1, 50,
39,
        9, 52, 34, 32, 40, 48,  5,  0,  0, 47, 56,  4, 24, 43, 46, 32,
42,
       28, 22, 40, 22, 49, 34, 16, 12, 51, 39, 40, 41,  0, 50,  0, 11,
56,
       60, 54, 11, 60, 38,  0, 19, 59,  7, 20, 17, 36, 12, 14,  3, 14,
61,
       49, 19, 31, 49, 36, 41, 41, 56, 29, 57, 31, 49, 58, 55, 16])
```

Decodifique estos para ver el texto predicho por este modelo no entrenado:

```
print("Input: \n", repr("".join(idx2char[input_example_batch[0]])))
print()
print("Next Char Predictions: \n",
repr("".join(idx2char[sampled_indices ])))
```

```
Input:
 "I'll groan, the way being short,\nAnd piece the way out with a heavy
heart.\nCome, come, in wooing sor"

Next Char Predictions:
 "mKY'hqJ \n$EZxMnskkopcMyLDjPwpPW
la3nVTbj'\n\nir&LehTdPJbJkVD?mabc\nl\n;rvp;vZ\nGu-
HEX?B$BwkGSkXccrQsSktqD"
```

- Entrenar al modelo:

En este punto, el problema puede tratarse como un problema de clasificación estándar. Dado el estado RNN anterior, y la entrada en este paso de tiempo, predice la clase del siguiente carácter.

Adjunte un optimizador y una función de pérdida.

La función estándar de pérdida tf.keras.losses.sparse_categorical_crossentropy funciona en este caso porque se aplica en la última dimensión de las predicciones.

Debido a que nuestro modelo devuelve logits, debemos establecer el indicador from_logits.

```
def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels,
logits, from_logits=True)
```

```
example_batch_loss  = loss(target_example_batch,
example_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape, " #
(batch_size, sequence_length, vocab_size)")
print("scalar_loss:      ", example_batch_loss.numpy().mean())
```

```
Prediction shape:  (64, 100, 65)  # (batch_size, sequence_length,
vocab_size)
scalar_loss:       4.1748996
```

Configure el procedimiento de capacitación utilizando el método tf.keras.Model.compile. Usaremos tf.keras.optimizers.Adam con argumentos predeterminados y la función de pérdida.

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

- Configurar puntos de control.

Use un tf.keras.callbacks.ModelCheckpoint para asegurarse de que los puntos de control se guarden durante el entrenamiento:

```
# Directory where the checkpoints will be saved
checkpoint_dir = './training_checkpoints'
# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

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

- Ejecute el entrenamiento:

Para mantener el tiempo de entrenamiento razonable, use 10 spochs para entrenar el modelo. En Colab, configure el tiempo de ejecución en GPU para un entrenamiento más rápido.

```
EPOCHS=10

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

```
Train for 172 steps
Epoch 1/10
```

```
172/172 [==============================] - 7s 40ms/step - loss: 2.6448
Epoch 2/10
172/172 [==============================] - 6s 35ms/step - loss: 1.9463
Epoch 3/10
172/172 [==============================] - 6s 34ms/step - loss: 1.6828
Epoch 4/10
172/172 [==============================] - 6s 34ms/step - loss: 1.5384
Epoch 5/10
172/172 [==============================] - 6s 34ms/step - loss: 1.4516
Epoch 6/10
172/172 [==============================] - 6s 34ms/step - loss: 1.3929
Epoch 7/10
172/172 [==============================] - 6s 33ms/step - loss: 1.3476
Epoch 8/10
172/172 [==============================] - 6s 34ms/step - loss: 1.3103
Epoch 9/10
172/172 [==============================] - 6s 36ms/step - loss: 1.2751
Epoch 10/10
172/172 [==============================] - 6s 34ms/step - loss: 1.2434
```

- Generar texto.

Restaurar el último punto de control

Para mantener este paso de predicción simple, use un tamaño de lote de 1.

Debido a la forma en que se pasa el estado RNN de un paso a otro, el modelo solo acepta un tamaño de lote fijo una vez construido.

Para ejecutar el modelo con un tamaño de lote diferente, necesitamos reconstruir el modelo y restaurar los pesos desde el punto de control.

```
tf.train.latest_checkpoint(checkpoint_dir)
```

```
'./training_checkpoints/ckpt_10'
```

```
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]))

model.summary()
```

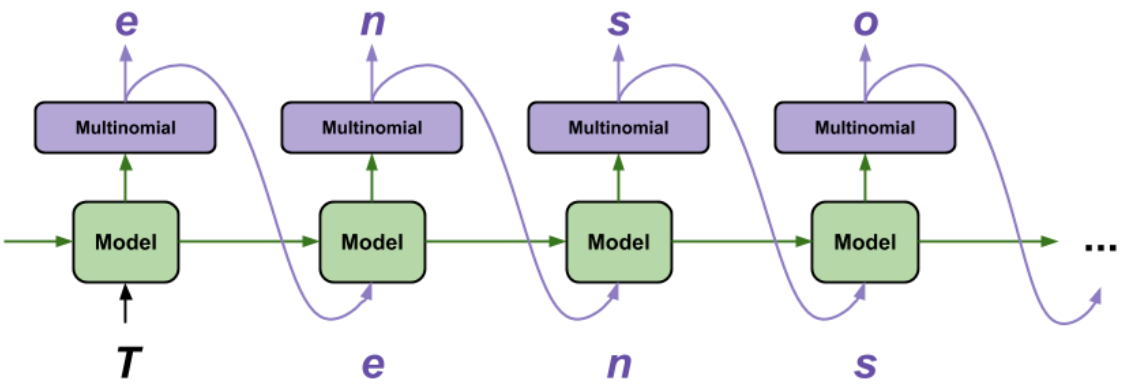
```
Model: "sequential_1"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (1, None, 256)            16640
_________________________________________________________________
gru_1 (GRU)                  (1, None, 1024)           3938304
_________________________________________________________________
dense_1 (Dense)              (1, None, 65)             66625
=================================================================
Total params: 4,021,569
Trainable params: 4,021,569
Non-trainable params: 0
_________________________________________________________________
```

El bucle de predicción

El siguiente bloque de código genera el texto:

- Comienza eligiendo una cadena de inicio, inicializando el estado RNN y configurando el número de caracteres a generar.
- Obtenga la distribución de predicción del siguiente carácter utilizando la cadena de inicio y el estado RNN.
- Luego, use una distribución categórica para calcular el índice del carácter predicho. Use este personaje predicho como nuestra próxima entrada al modelo.
- El estado RNN devuelto por el modelo se retroalimenta al modelo para que ahora tenga más contexto, en lugar de solo un carácter. Después de predecir el siguiente carácter, los estados RNN modificados se retroalimentan nuevamente en el modelo, que es cómo aprende a medida que obtiene más contexto de los caracteres predichos previamente.



Al observar el texto generado, verá que el modelo sabe cuándo capitalizar, hacer párrafos e imita un vocabulario de escritura similar a Shakespeare. Con el pequeño número de épocas de entrenamiento, aún no ha aprendido a formar oraciones coherentes.

```
def generate_text(model, start_string):
  # Evaluation step (generating text using the learned model)

  # Number of characters to generate
  num_generate = 1000

  # Converting our start string to numbers (vectorizing)
  input_eval = [char2idx[s] for s in start_string]
  input_eval = tf.expand_dims(input_eval, 0)

  # Empty string to store our results
  text_generated = []

  # Low temperatures results in more predictable text.
  # Higher temperatures results in more surprising text.
  # Experiment to find the best setting.
  temperature = 1.0

  # Here batch size == 1
  model.reset_states()
  for i in range(num_generate):
      predictions = model(input_eval)
      # remove the batch dimension
      predictions = tf.squeeze(predictions, 0)

      # using a categorical distribution to predict the character
returned by the model
      predictions = predictions / temperature
      predicted_id = tf.random.categorical(predictions,
num_samples=1)[-1,0].numpy()

      # We pass the predicted character as the next input to the model
      # along with the previous hidden state
      input_eval = tf.expand_dims([predicted_id], 0)

      text_generated.append(idx2char[predicted_id])

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

```
print(generate_text(model, start_string=u"ROMEO: "))
```

```
ROMEO: I am unplume, shalt to the
Francies would women cluncime against it.

MENENIUS:
Pemprescorce that you shall not be thus; let's in justices and by
have thoughts, to test the stars as great
As open lack gawned raged
Duke of Northumberland, this manner of his prince.

MARCIUS:
How do thou wast forced;
The endempily east enought than whence, or, bear
headed me aple, to-morrow why I rue,
My own brothers on't, but stins abbooon of
so sours; or ghinf purnicy in base as as
Two kings at my heart?

DUKE OF AUMERLE:
Which dost thou be in this person? peace: but whoreson hat a not flee
Whose honour and athe stampet's presence in sorrow is there?

BIONDELLO:
What can you gone, and suck'd your prunishen.
They, we beseech you, sister, whilst Margaret pleasant, gentle speed?
About! following proclaim the end,
And whom unto his rooate passion,
Go apperial court.
Now, sir, but 'tain for Richard, whom is that all.

PETRUCHIO:
Now, there, i' for the !
With welt repared on Rome, to
Grimy, sad I fought. Whe
```

Lo más fácil que puedes hacer para mejorar los resultados es entrenarlo por más tiempo (prueba EPOCHS = 30).

También puede experimentar con una cadena de inicio diferente, o intentar agregar otra capa RNN para mejorar la precisión del modelo, o ajustar el parámetro de temperatura para generar predicciones más o menos aleatorias.

- Avanzado: entrenamiento personalizado

El procedimiento de entrenamiento anterior es simple, pero no le da mucho control.

Entonces, ahora que ha visto cómo ejecutar el modelo manualmente, descomprimimos el ciclo de entrenamiento e implementémoslo nosotros mismos. Esto proporciona un punto de partida si, por ejemplo, se implementa el aprendizaje curricular para ayudar a estabilizar la salida de bucle abierto del modelo.

Usaremos tf.GradientTape para rastrear los gradientes. Puede obtener más información sobre este enfoque leyendo la ansiosa guía de ejecución.

El procedimiento funciona de la siguiente manera:

- Primero, inicialice el estado RNN. Hacemos esto llamando al método tf.keras.Model.reset_states.
- Luego, repita el conjunto de datos (lote por lote) y calcule las predicciones asociadas con cada uno.
- Abra un tf.GradientTape y calcule las predicciones y pérdidas en ese contexto.
- Calcule los gradientes de la pérdida con respecto a las variables del modelo utilizando el método tf.GradientTape.grads.

Finalmente, dé un paso hacia abajo utilizando el método f.train.Optimizer.apply_gradients del optimizador.

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

```
optimizer = tf.keras.optimizers.Adam()

@tf.function
def train_step(inp, target):
  with tf.GradientTape() as tape:
    predictions = model(inp)
    loss = tf.reduce_mean(
        tf.keras.losses.sparse_categorical_crossentropy(
            target, predictions, from_logits=True))
  grads = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(grads, model.trainable_variables))

  return loss
```

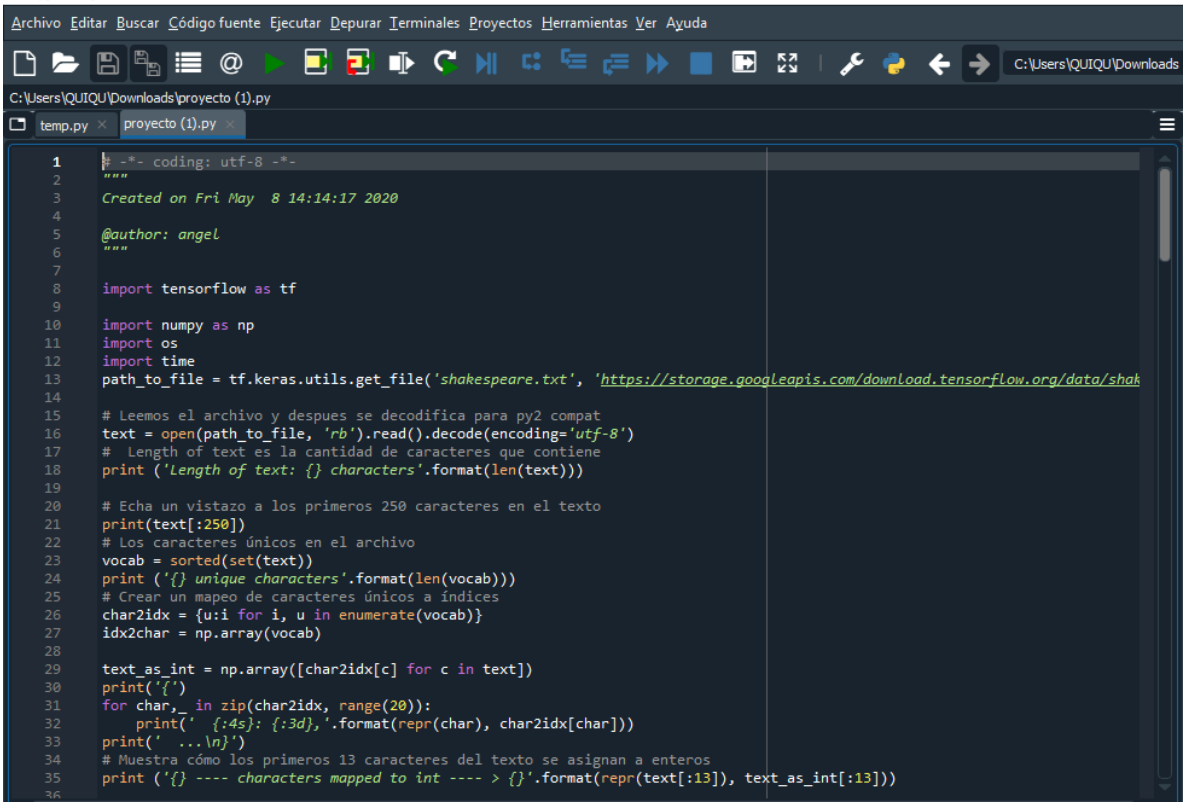
```
# Training step
EPOCHS = 10

for epoch in range(EPOCHS):
  start = time.time()

  # initializing the hidden state at the start of every epoch
  # initally hidden is None
  hidden = model.reset_states()

  for (batch_n, (inp, target)) in enumerate(dataset):
    loss = train_step(inp, target)

    if batch_n % 100 == 0:
      template = 'Epoch {} Batch {} Loss {}'
      print(template.format(epoch+1, batch_n, loss))

  # saving (checkpoint) the model every 5 epochs
  if (epoch + 1) % 5 == 0:
    model.save_weights(checkpoint_prefix.format(epoch=epoch))

  print ('Epoch {} Loss {:.4f}'.format(epoch+1, loss))
  print ('Time taken for 1 epoch {} sec\n'.format(time.time() -
start))

model.save_weights(checkpoint_prefix.format(epoch=epoch))
```

```
Epoch 1 Batch 0 Loss 4.175037384033203
Epoch 1 Batch 100 Loss 2.369581699371338
Epoch 1 Loss 2.1165
Time taken for 1 epoch 6.417382717132568 sec

Epoch 2 Batch 0 Loss 2.1537587642669678
Epoch 2 Batch 100 Loss 1.9563190937042236
Epoch 2 Loss 1.8062
Time taken for 1 epoch 5.329235792160034 sec

Epoch 3 Batch 0 Loss 1.8056563138961792
Epoch 3 Batch 100 Loss 1.7106741666793823
Epoch 3 Loss 1.6115
Time taken for 1 epoch 5.339670419692993 sec
```

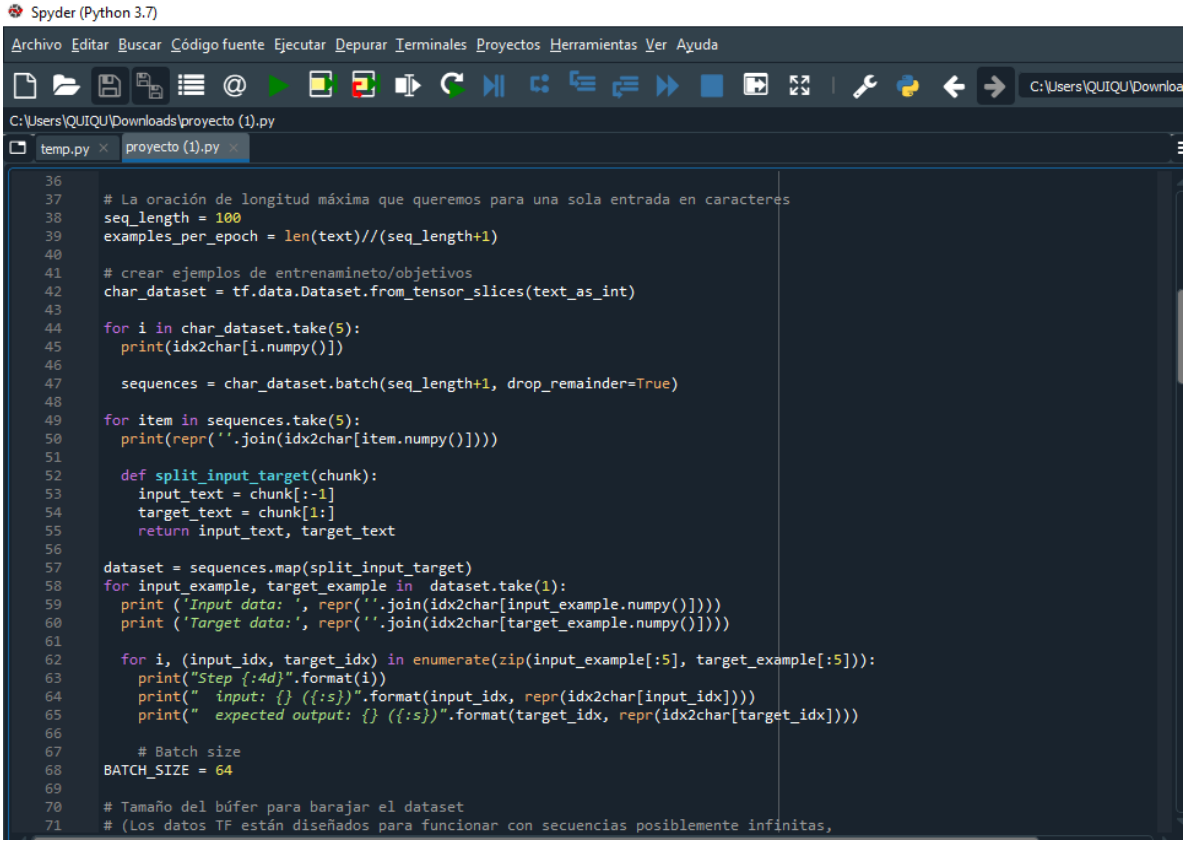
```
Epoch 4 Batch 0 Loss 1.5668939352035522
Epoch 4 Batch 100 Loss 1.5268672704696655
Epoch 4 Loss 1.4890
Time taken for 1 epoch 5.412369728088379 sec

Epoch 5 Batch 0 Loss 1.4938063621520996
Epoch 5 Batch 100 Loss 1.4473059177398682
Epoch 5 Loss 1.4000
Time taken for 1 epoch 5.519220590591431 sec

Epoch 6 Batch 0 Loss 1.3862831592559814
Epoch 6 Batch 100 Loss 1.410801887512207
Epoch 6 Loss 1.3671
Time taken for 1 epoch 5.347445011138916 sec

Epoch 7 Batch 0 Loss 1.346337080001831
Epoch 7 Batch 100 Loss 1.3436977863311768
Epoch 7 Loss 1.3429
Time taken for 1 epoch 5.457265853881836 sec

Epoch 8 Batch 0 Loss 1.3194979429244995
Epoch 8 Batch 100 Loss 1.3179987668991089
Epoch 8 Loss 1.3036
Time taken for 1 epoch 5.361689329147339 sec

Epoch 9 Batch 0 Loss 1.253419041633606
Epoch 9 Batch 100 Loss 1.2753329277038574
Epoch 9 Loss 1.2893
Time taken for 1 epoch 5.390798807144165 sec

Epoch 10 Batch 0 Loss 1.1887139081954956
Epoch 10 Batch 100 Loss 1.2455520629882812
Epoch 10 Loss 1.2771
Time taken for 1 epoch 5.569040298461914 sec
```

CÓDIGO ELABORADO EN SPYDER:

Spyder (Python 3.7)



Spyder (Python 3.7)



Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda

C:\Users\QUIQU\Downloads\proyecto (1).py

temp.py | proyecto (1).py

```
# Tamaño del búfer para barajar el dataset
# (Los datos TF están diseñados para funcionar con secuencias posiblemente infinitas,
# para que no intente barajar toda la secuencia en la memoria. En cambio,mantiene un búfer en el que baraja elementos).
BUFFER_SIZE = 10000

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

dataset

# Longitud del vocabulario en caracteres
vocab_size = len(vocab)

# La dimensión de incrustación
embedding_dim = 256

# Número de unidades RNN
rnn_units = 1024

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.GRU(rnn_units,
                        return_sequences=True,
                        stateful=True,
                        recurrent_initializer='glorot_uniform'),
    tf.keras.layers.Dense(vocab_size)
  ])
  return model

model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)

```

Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda

C:\Users\QUIQU\Downloads\proyecto (1).py

temp.py | proyecto (1).py

```

for input_example_batch, target_example_batch in dataset.take(1):
  example_batch_predictions = model(input_example_batch)
  print(example_batch_predictions.shape, "# (batch_size, sequence_length, vocab_size)")

  model.summary()

  sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
sampled_indices = tf.squeeze(sampled_indices,axis=-1).numpy()
sampled_indices

print("Input: \n", repr("".join(idx2char[input_example_batch[0]])))
print()
print("Next Char Predictions: \n", repr("".join(idx2char[sampled_indices ])))

def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

example_batch_loss  = loss(target_example_batch, example_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, sequence_length, vocab_size)")
print("scalar_loss:      ", example_batch_loss.numpy().mean())
model.compile(optimizer='adam', loss=loss)
# directorio donde se guardaran los checkpints
checkpoint_dir = './training_checkpoints'
# Nombre de los archivos chekpoint
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

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

EPOCHS=10
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
tf.train.latest_checkpoint(checkpoint_dir)
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)

```

Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda

C:\Users\QUIQU\Downloads\proyecto (1).py

temp.py | proyecto (1).py

```
140
141   model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
142
143   model.build(tf.TensorShape([1, None]))
144   model.summary()
145   def generate_text(model, start_string):
146    # Paso de evaluación (generar texto usando el modelo aprendido)
147
148     # Número de caracteres a generar
149     num_generate = 1000
150
151     # Convertir nuestra cadena de inicio a números (vectorización)
152     input_eval = [char2idx[s] for s in start_string]
153     input_eval = tf.expand_dims(input_eval, 0)
154
155     # Cadena vacía para almacenar nuestros resultados.
156     text_generated = []
157
158      # Las bajas temperaturas dan como resultado un texto más predecible.Temperaturas más altas resultan en texto más sorpren
159     temperature = 1.0
160
161     # batch size == 1
162     model.reset_states()
163     for i in range(num_generate):
164         predictions = model(input_eval)
165         # eliminar la dimensión del batch
166         predictions = tf.squeeze(predictions, 0)
167
168        # usamos una distribución categórica para predecir el carácter devuelto por el modelo
169         predictions = predictions / temperature
170         predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()
171
172         # Pasamos el carácter predicho como la siguiente entrada al modelo,junto con el estado oculto anterior
173         input_eval = tf.expand_dims([predicted_id], 0)
174
175         text_generated.append(idx2char[predicted_id])
```

Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda

C:\Users\QUIQU\Downloads\proyecto (1).py

temp.py | proyecto (1).py

```
177     return (start_string + ''.join(text_generated))
178
179   print(generate_text(model, start_string=u"ROMEO: "))
180   model = build_model(
181     vocab_size = len(vocab),
182     embedding_dim=embedding_dim,
183     rnn_units=rnn_units,
184     batch_size=BATCH_SIZE)
185   optimizer = tf.keras.optimizers.Adam()
186
187   @tf.function
188   def train_step(inp, target):
189     with tf.GradientTape() as tape:
190       predictions = model(inp)
191       loss = tf.reduce_mean(
192           tf.keras.losses.sparse_categorical_crossentropy(
193               target, predictions, from_logits=True))
194     grads = tape.gradient(loss, model.trainable_variables)
195     optimizer.apply_gradients(zip(grads, model.trainable_variables))
196
197     return loss
198   # Pasos de entrenamiendo
199   EPOCHS = 10
200
201   for epoch in range(EPOCHS):
202     start = time.time()
203
204    # inicializando el estado oculto al comienzo de cada epoch,Inicialmente oculto es Ninguno
205     hidden = model.reset_states()
206
207     for (batch_n, (inp, target)) in enumerate(dataset):
208       loss = train_step(inp, target)
209
210       if batch_n % 100 == 0:
211         template = 'Epoch {} Batch {} Loss {}'
212         print(template.format(epoch+1, batch_n, loss))
```

Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda

C:\Users\QUIQU\Downloads\proyecto (1).py

temp.py | proyecto (1).py

```
      predictions = model(inp)
      loss = tf.reduce_mean(
          tf.keras.losses.sparse_categorical_crossentropy(
              target, predictions, from_logits=True))
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    return loss
# Pasos de entrenamiendo
EPOCHS = 10

for epoch in range(EPOCHS):
  start = time.time()

 # inicializando el estado oculto al comienzo de cada epoch,Inicialmente oculto es Ninguno
  hidden = model.reset_states()

  for (batch_n, (inp, target)) in enumerate(dataset):
    loss = train_step(inp, target)

    if batch_n % 100 == 0:
      template = 'Epoch {} Batch {} Loss {}'
      print(template.format(epoch+1, batch_n, loss))

  # saving (checkpoint) the model every 5 epochs
  # guardardamos (chekpoint) el modelo cada 5 epochs
  if (epoch + 1) % 5 == 0:
    model.save_weights(checkpoint_prefix.format(epoch=epoch))

  print ('Epoch {} Loss {:.4f}'.format(epoch+1, loss))
  print ('Time taken for 1 epoch {} sec\n'.format(time.time() - start))

model.save_weights(checkpoint_prefix.format(epoch=epoch))
```

RESULTADO DE LA EJECUCIÓN DEL CODIGO:

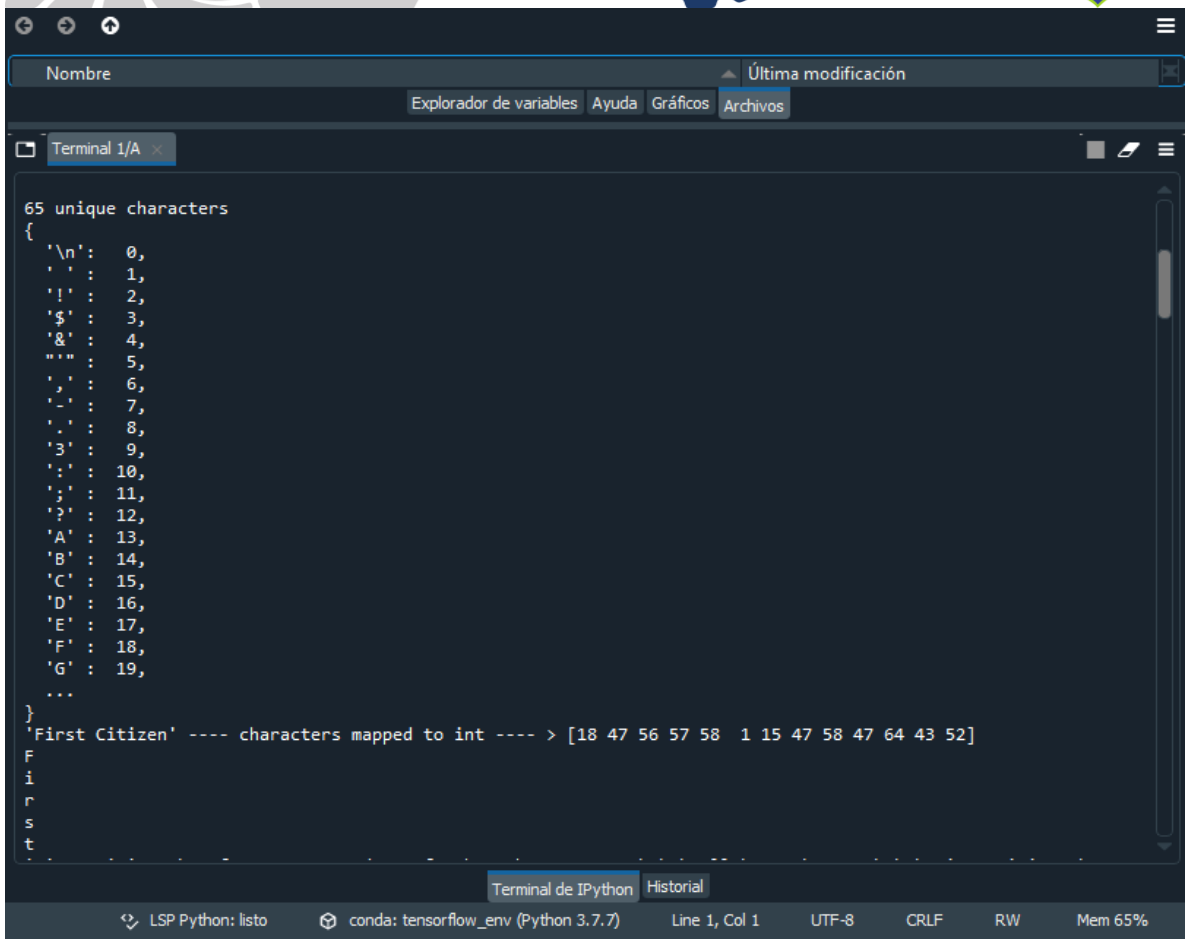
Nombre | Última modificación

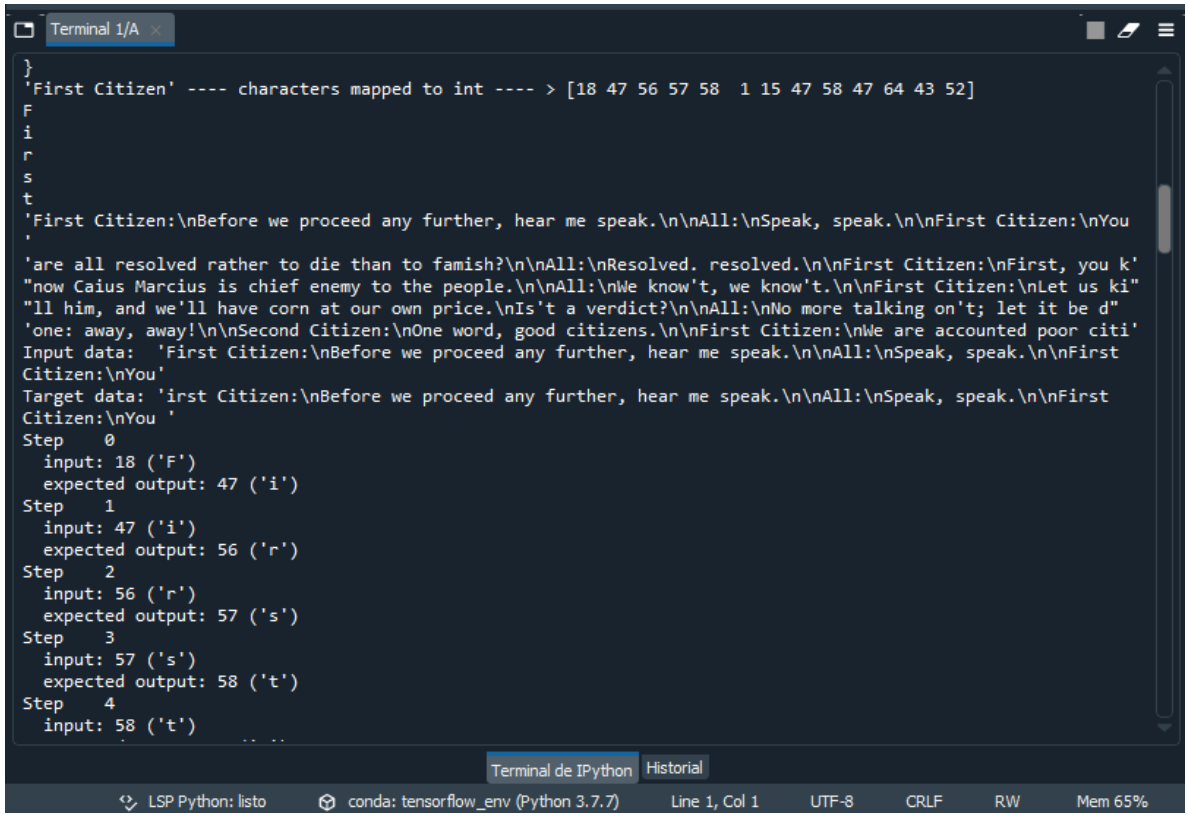
Explorador de variables | Ayuda | Gráficos | Archivos

Terminal 1/A

```
Python 3.7.7 (default, May  6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/QUIQU/Downloads/proyecto (1).py', wdir='C:/Users/QUIQU/Downloads')
Length of text: 1115394 characters
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

65 unique characters
{
  '\n':   0,
  ' ' :   1,
  '!' :   2,
  '$' :   3,
  '&' :   4,
  "'" :   5,
  ',' :   6,
```

Terminal de IPython | Historial


Nombre
Última modificación
Explorador de variables
Ayuda
Gráficos
Archivos
Terminal 1/A
65 unique characters
{
'\n': 0,
' ' : 1,
'!' : 2,
'$' : 3,
'&' : 4,
"'" : 5,
',' : 6,
'-' : 7,
'.' : 8,
'3' : 9,
':' : 10,
';' : 11,
'?' : 12,
'A' : 13,
'B' : 14,
'C' : 15,
'D' : 16,
'E' : 17,
'F' : 18,
'G' : 19,
...
}
'First Citizen' ---- characters mapped to int ---- > [18 47 56 57 58 1 15 47 58 47 64 43 52]
F
i
r
s
t
Terminal de IPython
Historial
LSP Python: listo
conda: tensorflow_env (Python 3.7.7)
Line 1, Col 1
UTF-8
CRLF
RW
Mem 65%


Terminal 1/A
}
'First Citizen' ---- characters mapped to int ---- > [18 47 56 57 58 1 15 47 58 47 64 43 52]
F
i
r
s
t
'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou
'
'are all resolved rather to die than to famish?\n\nAll:\nResolved. resolved.\n\nFirst Citizen:\nFirst, you k'
"now Caius Marcius is chief enemy to the people.\n\nAll:\nWe know't, we know't.\n\nFirst Citizen:\nLet us ki"
"ll him, and we'll have corn at our own price.\nIs't a verdict?\n\nAll:\nNo more talking on't; let it be d"
'one: away, away!\n\nSecond Citizen:\nOne word, good citizens.\n\nFirst Citizen:\nWe are accounted poor citi'
Input data: 'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst
Citizen:\nYou'
Target data: 'irst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst
Citizen:\nYou '
Step 0
input: 18 ('F')
expected output: 47 ('i')
Step 1
input: 47 ('i')
expected output: 56 ('r')
Step 2
input: 56 ('r')
expected output: 57 ('s')
Step 3
input: 57 ('s')
expected output: 58 ('t')
Step 4
input: 58 ('t')
Terminal de IPython
Historial
LSP Python: listo
conda: tensorflow_env (Python 3.7.7)
Line 1, Col 1
UTF-8
CRLF
RW
Mem 65%

Terminal 1/A

```
  expected output: 58 ('t')
Step    4
  input: 58 ('t')
  expected output: 1 (' ')
(64, 100, 65) # (batch_size, sequence_length, vocab_size)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (64, None, 256)           16640
_________________________________________________________________
gru (GRU)                    (64, None, 1024)          3938304
_________________________________________________________________
dense (Dense)                (64, None, 65)            66625
=================================================================
Total params: 4,021,569
Trainable params: 4,021,569
Non-trainable params: 0
_________________________________________________________________
Input:
 "e;\nThe crown, usurp'd, disgraced his kingly glory.\nif something thou wilt swear to be believed,\nSwea"

Next Char Predictions:
 "\n ;kDowdrIruHkDxMUXU'YBOvYXQnroA-zZbMc,FWnXWaHoywPJkPE3kUartGceTjAtmgEDl\nf-eMZV\nU.\nh:YSCl:aHZkaU,sPM"
Prediction shape:  (64, 100, 65)  # (batch_size, sequence_length, vocab_size)
scalar_loss:       4.1739354
Train for 172 steps
Epoch 1/10
172/172 [==============================] - 3023s 18s/step - loss: 2.6894
Epoch 2/10
172/172 [==============================] - 3037s 18s/step - loss: 1.9495
```

Nombre | Última modificación

Explorador de variables | Ayuda | Gráficos | Archivos

Terminal 1/A

```
Epoch 3/10
172/172 [==============================] - 3044s 18s/step - loss: 1.6847
Epoch 4/10
172/172 [==============================] - 3046s 18s/step - loss: 1.5396
Epoch 5/10
172/172 [==============================] - 2974s 17s/step - loss: 1.4536
Epoch 6/10
172/172 [==============================] - 3274s 19s/step - loss: 1.3954
Epoch 7/10
172/172 [==============================] - 3503s 20s/step - loss: 1.3495
Epoch 8/10
172/172 [==============================] - 3679s 21s/step - loss: 1.3130
Epoch 9/10
172/172 [==============================] - 3632s 21s/step - loss: 1.2785
Epoch 10/10
172/172 [==============================] - 3143s 18s/step - loss: 1.2467
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (1, None, 256)            16640
_________________________________________________________________
gru_1 (GRU)                  (1, None, 1024)           3938304
_________________________________________________________________
dense_1 (Dense)              (1, None, 65)             66625
=================================================================
Total params: 4,021,569
Trainable params: 4,021,569
Non-trainable params: 0
_________________________________________________________________
ROMEO: he slaid I'll wear myself?
```

Terminal 1/A

```
ROMEO: he slaid I'll wear myself?
We wish me than I was; my lord: it is broke, dead yours wish we dead, you
too whoo that his swoft assore.

MISTRESS OVERDONEO:
What, ho! no, since you?

First Citizen:
Resue, thy father hath guints not assembly.

DUKE OF YORK:
Blad my dukedom will bear the king:
For whorged! what's the law, quickly, mount, good nurse, my good Lady.

CAMILLO:
He would be obedience against his hand-shall
Stand night's credul and Havingham
And began to grite your will. What we inclined attend? his badgempent
And cit when ever he shall speak broke
And wrong it, sweet loverst: his condicience,
And twenty tell what's those I less the malthat flottress in heaven!
Lovely and by the house is alight, and pilved his hairs;
For himseyfit thing paralls of the plencetory, like a lich
Than Lord Hastings that know your lovest, to your came;
Fie! yet she say age.

AUFIDIUS:
What is the glos of their be loves,
My word compary, since you have lost it, company, and Henry, godours sove forfeitnes
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer
```

Terminal 1/A

```
AUFIDIUS:
What is the glos of their be loves,
My word compary, since you have lost it, company, and Henry, godours sove forfeitnes
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_1
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_2
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm' for
(root).layer_with_weights-0.embeddings
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm' for
(root).layer_with_weights-2.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm' for
(root).layer_with_weights-2.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm' for
(root).layer_with_weights-1.cell.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm' for
(root).layer_with_weights-1.cell.recurrent_kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm' for
(root).layer_with_weights-1.cell.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v' for
(root).layer_with_weights-0.embeddings
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v' for
(root).layer_with_weights-2.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v' for
(root).layer_with_weights-2.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v' for
(root).layer_with_weights-1.cell.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v' for
(root).layer_with_weights-1.cell.recurrent_kernel
```

Terminal 1/A

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v' for
(root).layer_with_weights-1.cell.bias
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore or
tf.keras.Model.load_weights) but not all checkpointed values were used. See above for specific issues. Use
expect_partial() on the load status object, e.g. tf.train.Checkpoint.restore(...).expect_partial(), to
silence these warnings, or use assert_consumed() to make the check explicit. See https://www.tensorflow.org/
guide/checkpoint#loading_mechanics for details.
Epoch 1 Batch 0 Loss 4.174539566040039
Epoch 1 Batch 100 Loss 2.341681718826294
Epoch 1 Loss 2.1440
Time taken for 1 epoch 3295.1990463733673 sec

Epoch 2 Batch 0 Loss 2.1683223247528076
Epoch 2 Batch 100 Loss 1.8864773511886597
Epoch 2 Loss 1.8120
Time taken for 1 epoch 3900.7975373268127 sec

Epoch 3 Batch 0 Loss 1.7885384559631348
Epoch 3 Batch 100 Loss 1.6584231853485107
Epoch 3 Loss 1.6008
Time taken for 1 epoch 3643.7905201911926 sec

Epoch 4 Batch 0 Loss 1.6050207614898682
Epoch 4 Batch 100 Loss 1.529829502105713
Epoch 4 Loss 1.4708
Time taken for 1 epoch 4459.008887052536 sec

Epoch 5 Batch 0 Loss 1.472581148147583
Epoch 5 Batch 100 Loss 1.4854211807250977
Epoch 5 Loss 1.4134
Time taken for 1 epoch 2927.036178588867 sec
```

Terminal 1/A

```
Epoch 6 Batch 0 Loss 1.3844778537750244
Epoch 6 Batch 100 Loss 1.3778408765792847
Epoch 6 Loss 1.3326
Time taken for 1 epoch 2949.542879343033 sec

Epoch 7 Batch 0 Loss 1.3346383571624756
Epoch 7 Batch 100 Loss 1.3786982297897339
Epoch 7 Loss 1.3648
Time taken for 1 epoch 2888.9217858314514 sec

Epoch 8 Batch 0 Loss 1.3013850450515747
Epoch 8 Batch 100 Loss 1.3075834512710571
Epoch 8 Loss 1.3470
Time taken for 1 epoch 2861.0752654075623 sec

Epoch 9 Batch 0 Loss 1.2709742784500122
Epoch 9 Batch 100 Loss 1.3190407752990723
Epoch 9 Loss 1.3004
Time taken for 1 epoch 2839.4306960105896 sec

Epoch 10 Batch 0 Loss 1.2163399457931519
Epoch 10 Batch 100 Loss 1.2429767847061157
Epoch 10 Loss 1.2404
Time taken for 1 epoch 2834.1162688732147 sec


In [2]:
```

Conclusiones.

En definitiva, el Machine Learning es un maestro del reconocimiento de patrones, y es capaz de convertir una muestra de datos en un programa informático capaz de extraer inferencias de nuevos conjuntos de datos para los que no ha sido entrenado previamente.

Esta capacidad de aprendizaje se emplea también para la mejora de motores de búsqueda, la robótica, el diagnóstico médico o incluso la detección del fraude en el uso de tarjetas de crédito.

La estadística es sin duda la base fundamental del aprendizaje automático, que básicamente consiste en una serie de algoritmos capaces de analizar una serie de algoritmos capaces de analizar grandes cantidades de datos para deducir cual es el resultado más óptimo para un determinado problema.

Bibliografía.

- https://stackabuse.com/text-generation-with-python-and-tensorflow-keras/
- https://keras.io/examples/generative/lstm_character_level_text_generation/
- https://towardsdatascience.com/text-generation-using-rnns-fdb03a010b9f
- https://gilberttanner.com/blog/generating-text-using-a-recurrent-neuralnetwork
- http://www.gutenberg.org/
- https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/