Actividad - Redes Neuronales Profundas

November 7, 2023

1 Actividad: Redes Neuronales Profundas

Alfonso Pineda Cedillo | A01660394

Fecha de entrega: 7 de Noviembre de 2023

Instrucciones:

- Deberás de entrenar la red utilizando un libro en formato txt del tema de tu elección.
- Muestra capturas de pantalla con resultados de 2 entradas y 3 temperaturas diferentes (6 resultados en total)
- Sube un pdf donde muestres tu código y los resultados, además de mostrar el nombre o tema del libro utilizado para entrenar la red.

1.1 Solución

En primer instancia, importamos las librerías necesarias para el desarrollo de la actividad.

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

1.1.1 Importación de texto de entrenamiento

Para la actividad actual, se seleccionó el libro "A Game of Thrones" de George R. R. Martin, el cual se encuentra en formato txt. Para importar el texto, se utilizó la librería `urlib` y se guardó en una variable llamada `text`.

[2]:
```
import urllib.request

url = 'https://raw.githubusercontent.com/nihitx/game-of-thrones-/master/
 ↪gameofthrones.txt'
path_to_file = 'gameofthrones.txt'  # Nombre del archivo local en el que se␣
 ↪guardará el texto descargado

# Descargar el archivo de la URL
urllib.request.urlretrieve(url, path_to_file)
```

```
# Leer el texto del archivo descargado
with open(path_to_file, 'r', encoding='utf-8') as file:
    text = file.read()

print('Longitud del texto: {} caracteres'.format(len(text)))

vocab = sorted(set(text))  # Vocabulario (todos los caracteres diferentes)
print('Caracteres diferentes: {}'.format(len(vocab)))
print('Vocabulario: ', vocab)
```

```
Longitud del texto: 5662324 caracteres
Caracteres diferentes: 86
Vocabulario:  ['\n', ' ', '!', '(', ')', ',', '-', '.', '/', '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', 'ê', '–', '‘', '’', '"', '"', '…']
```

1.1.2 Tokenización e inversión del vocabulario

[3]:
```
char2idx = {u:i for i, u in enumerate(vocab)} # diccionario que asigna un␣
 ↪índice a cada caracter

idx2char = np.array(vocab) # array que asigna un caracter a cada índice (el␣
 ↪inverso del anterior)

for char, _ in zip(char2idx, range(len(vocab))):
    print('{:6s} ---> {:4d}'.format(repr(char), char2idx[char]))
```

```
'\n'   --->    0
' '    --->    1
'!'    --->    2
'('    --->    3
')'    --->    4
','    --->    5
'-'    --->    6
'.'    --->    7
'/'    --->    8
'0'    --->    9
'1'    --->   10
'2'    --->   11
'3'    --->   12
'4'    --->   13
'5'    --->   14
'6'    --->   15
```

```
'7'    --->   16
'8'    --->   17
'9'    --->   18
':'    --->   19
';'    --->   20
'?'    --->   21
'A'    --->   22
'B'    --->   23
'C'    --->   24
'D'    --->   25
'E'    --->   26
'F'    --->   27
'G'    --->   28
'H'    --->   29
'I'    --->   30
'J'    --->   31
'K'    --->   32
'L'    --->   33
'M'    --->   34
'N'    --->   35
'O'    --->   36
'P'    --->   37
'Q'    --->   38
'R'    --->   39
'S'    --->   40
'T'    --->   41
'U'    --->   42
'V'    --->   43
'W'    --->   44
'X'    --->   45
'Y'    --->   46
'Z'    --->   47
'['    --->   48
']'    --->   49
'a'    --->   50
'b'    --->   51
'c'    --->   52
'd'    --->   53
'e'    --->   54
'f'    --->   55
'g'    --->   56
'h'    --->   57
'i'    --->   58
'j'    --->   59
'k'    --->   60
'l'    --->   61
'm'    --->   62
'n'    --->   63
```

```
'o'    --->   64
'p'    --->   65
'q'    --->   66
'r'    --->   67
's'    --->   68
't'    --->   69
'u'    --->   70
'v'    --->   71
'w'    --->   72
'x'    --->   73
'y'    --->   74
'z'    --->   75
'{'    --->   76
'}'    --->   77
'é'    --->   78
'ê'    --->   79
'-'    --->   80
'‘'    --->   81
'’'    --->   82
'"'    --->   83
'"'    --->   84
'…'    --->   85
```

1.1.3 Conversión de texto a secuencias de enteros

[4]:
```
text_as_int = np.array([char2idx[c] for c in text]) # array que asigna un␣
 ↪índice a cada caracter del texto

# Mostramos cómo se ha codificado el texto
print ('Texto: ', format(repr(text[:50])), '...'
       '\nCodificado: ', text_as_int[:50])
```

```
Texto:  '\n\n"We should start back," Gared urged as the woods' …
Codificado:  [ 0  0 83 44 54  1 68 57 64 70 61 53  1 68 69 50 67 69  1 51 50 52
60  5
 84  1 28 50 67 54 53  1 70 67 56 54 53  1 50 68  1 69 57 54  1 72 64 64
 53 68]
```

1.1.4 Preparación de los datos de entrenamiento y prueba

[5]:
```
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int) # creamos un␣
 ↪dataset con los índices del texto (texto codificado)

seq_length = 100 # longitud de la secuencia de entrada (número de caracteres␣
 ↪que se le pasan a la red)
```

```
sequences = char_dataset.batch(seq_length + 1, drop_remainder=True) # dividimos el dataset en secuencias de longitud seq_length+1 porque la red predice el siguiente caracter a partir de los seq_length anteriores
```

[6]:
```
# Comprobación de que se ha dividido correctamente
for item in sequences.take(5):
  print(repr(''.join(idx2char[item.numpy()])))
```

```
'\n\n"We should start back," Gared urged as the woods began to grow dark around
them. "The wildlings are'
' dead."\n\n"Do the dead frighten you?" Ser Waymar Royce asked with just the
hint of a smile.\n\nGared did'
' not rise to the bait. He was an old man, past fifty, and he had seen the
lordlings come and go. "Dea'
'd is dead," he said. "We have no business with the dead."\n\n"Are they dead?"
Royce asked softly. "What'
' proof have we?"\n\n"Will saw them," Gared said. "If he says they are dead,
that’s proof enough for me.'
```

[7]:
```
# Preparación de datos de entrenamiento
# La entrada corresponde al caracter 0 hasta el caracter 99
# La salida corresponde al caracter 1 hasta el caracter 100

def split_input_target(chunk): # función que separa la entrada de la salida
  input_text = chunk[:-1] # entrada
  target_text = chunk[1:] # salida
  return input_text, target_text

dataset = sequences.map(split_input_target) # aplicamos la función anterior a cada secuencia
```

[8]:
```
# Agrupamiento de los datos en lotes de tamaño 64

BATCH_SIZE = 64
BUFFER_SIZE = 10000   # Tamaño del buffer para barajar

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True) # Drop_remainder=True para que todos los lotes tengan el mismo tamaño, los lotes que no se alcancen a completar se descartan
print(dataset)
```

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

1.1.5 Definición del modelo

[9]:
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size): # función␣
 ↪que crea el modelo
  model = tf.keras.Sequential([
    layers.Embedding(vocab_size, embedding_dim, batch_input_shape=[batch_size,␣
 ↪None]), # capa de embedding
    layers.LSTM(rnn_units, return_sequences=True, stateful=True,␣
 ↪recurrent_initializer='glorot_uniform'), # capa LSTM
    layers.Dense(vocab_size) # capa densa
  ])
  return model
```

[10]:
```
vocab_size = len(vocab) # tamaño del vocabulario
embedding_dim = 256 # dimensión del embedding
rnn_units = 1024 # número de neuronas de la capa LSTM

model = build_model(vocab_size, embedding_dim, rnn_units, BATCH_SIZE) #␣
 ↪construimos el modelo

model.summary() # mostramos un resumen del modelo
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (64, None, 256)           22016

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

 dense (Dense)               (64, None, 86)            88150

=================================================================
Total params: 5357142 (20.44 MB)
Trainable params: 5357142 (20.44 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1.1.6 Entrenamiento del modelo

[11]:
```
def loss(labels, logits): # función de pérdida
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits,␣
 ↪from_logits=True)
```

[12]:
```
# Compilación del modelo
model.compile(optimizer='adam', loss=loss)
```

[13]:
```
# Añadir checkpoints para almacenar los pesos del modelo en cada época

checkpoint_dir = './training_checkpoints' # directorio donde se guardarán los␣
 ↪checkpoints
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}") # nombre de␣
 ↪los archivos de los checkpoints

checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
  filepath=checkpoint_prefix,
  save_weights_only=True) # sólo se guardan los pesos
```

[14]:
```
EPOCHS = 50 # número de épocas
model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback]) #␣
 ↪entrenamiento del modelo
```

```
Epoch 1/50
875/875 [==============================] - 71s 65ms/step - loss: 1.7547
Epoch 2/50
875/875 [==============================] - 60s 66ms/step - loss: 1.2940
Epoch 3/50
875/875 [==============================] - 66s 71ms/step - loss: 1.2137
Epoch 4/50
875/875 [==============================] - 64s 71ms/step - loss: 1.1709
Epoch 5/50
875/875 [==============================] - 64s 71ms/step - loss: 1.1408
Epoch 6/50
875/875 [==============================] - 64s 72ms/step - loss: 1.1162
Epoch 7/50
875/875 [==============================] - 65s 73ms/step - loss: 1.0956
Epoch 8/50
875/875 [==============================] - 65s 73ms/step - loss: 1.0774
Epoch 9/50
875/875 [==============================] - 65s 73ms/step - loss: 1.0610
Epoch 10/50
875/875 [==============================] - 65s 72ms/step - loss: 1.0462
Epoch 11/50
875/875 [==============================] - 64s 71ms/step - loss: 1.0326
Epoch 12/50
875/875 [==============================] - 64s 71ms/step - loss: 1.0204
Epoch 13/50
875/875 [==============================] - 64s 71ms/step - loss: 1.0093
Epoch 14/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9994
Epoch 15/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9905
Epoch 16/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9827
```

```
Epoch 17/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9756
Epoch 18/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9696
Epoch 19/50
875/875 [==============================] - 64s 72ms/step - loss: 0.9646
Epoch 20/50
875/875 [==============================] - 65s 72ms/step - loss: 0.9601
Epoch 21/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9565
Epoch 22/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9532
Epoch 23/50
875/875 [==============================] - 65s 72ms/step - loss: 0.9507
Epoch 24/50
875/875 [==============================] - 65s 72ms/step - loss: 0.9487
Epoch 25/50
875/875 [==============================] - 66s 73ms/step - loss: 0.9478
Epoch 26/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9473
Epoch 27/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9462
Epoch 28/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9464
Epoch 29/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9467
Epoch 30/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9481
Epoch 31/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9490
Epoch 32/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9507
Epoch 33/50
875/875 [==============================] - 66s 73ms/step - loss: 0.9529
Epoch 34/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9540
Epoch 35/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9564
Epoch 36/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9581
Epoch 37/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9624
Epoch 38/50
875/875 [==============================] - 66s 73ms/step - loss: 0.9638
Epoch 39/50
875/875 [==============================] - 64s 72ms/step - loss: 0.9670
Epoch 40/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9729
```

```
Epoch 41/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9763
Epoch 42/50
875/875 [==============================] - 64s 72ms/step - loss: 0.9800
Epoch 43/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9833
Epoch 44/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9865
Epoch 45/50
875/875 [==============================] - 64s 71ms/step - loss: 0.9936
Epoch 46/50
875/875 [==============================] - 65s 73ms/step - loss: 0.9975
Epoch 47/50
875/875 [==============================] - 66s 73ms/step - loss: 1.0035
Epoch 48/50
875/875 [==============================] - 64s 71ms/step - loss: 1.0103
Epoch 49/50
875/875 [==============================] - 66s 73ms/step - loss: 1.0147
Epoch 50/50
875/875 [==============================] - 64s 71ms/step - loss: 1.0194
```

[14]: `<keras.src.callbacks.History at 0x7e7fe24e6d40>`

1.1.7 Generación de texto

[15]:
```
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1) #␣
 ↪construimos el modelo con batch_size=1
model.load_weights(tf.train.latest_checkpoint(checkpoint_dir)) # cargamos los␣
 ↪pesos del último checkpoint

model.build(tf.TensorShape([1, None])) # construimos el modelo con␣
 ↪batch_size=1, el 1 es porque solo se espera 1 oración de entrada
```

[32]:
```
# Función para generar texto caracter a caracter

def generate_text(model, start_string):
  num_generate = 500 # número de caracteres a generar
  input_eval = [char2idx[s] for s in start_string] # convertimos la cadena de␣
 ↪entrada en una lista de índices (números)

  input_eval = tf.expand_dims(input_eval, 0) # añadimos una dimensión al␣
 ↪principio (batch_size=1)

  text_generated = [] # lista para almacenar el texto generado

  temperature = 0.9 # parámetro para controlar la aleatoriedad de la predicción␣
 ↪(0 = predicción determinista, 1 = predicción aleatoria)
```

```
  model.reset_states() # reiniciamos el estado de la red

  for i in range(num_generate):
    predictions = model(input_eval) # predicciones de la red
    predictions = tf.squeeze(predictions, 0)
    predictions = predictions / temperature  # dividimos entre temperature para␣
↪controlar la aleatoriedad de la predicción

    predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].
↪numpy()  # obtenemos el índice de la predicción (el caracter predicho)
                                                                              ␣
↪    # [-1,0] para obtener el último caracter predicho
    input_eval = tf.expand_dims([predicted_id], 0)

    text_generated.append(idx2char[predicted_id]) # añadimos el caracter␣
↪predicho a la lista

  return (start_string + ''.join(text_generated)) # devolvemos la cadena de␣
↪entrada + la cadena generada
```

1.1.8 Resultados

A continuación, analizaremos los resultados obtenidos en la generación de texto a partir de la cadena "Dragon" y una temperatura de 0.1.

[30]:
```
# Temperatura = 0.1
print(generate_text(model, start_string="Dragon ")) # generamos texto a partir␣
 ↪de la cadena "Dragon "
```

```
Dragon before he was so stupid as the others said. "I will not see the shape of
your hands."

"I will not have to say that the sea of the water was stupid." The old man shook
his head. "I was a knight before I could find you."

"I would have to say that the girl was not about to say that you were a strong
power to come to me."

"I will not have to say that the girl who was the best thing I have now. I was a
knight, and the stars won't be crowned and well as she was a man of the Night's
Watch. I was the
```

Probamos el mismo input con una temperatura de 0.5.

[24]:
```
# Temperatura = 0.5
print(generate_text(model, start_string="Dragon ")) # generamos texto a partir␣
 ↪de la cadena "Dragon "
```

```
Dragon showed a muddy drink of white scat. "The castle now," he said, "but he
was the bloody banner."

"We have to go to your castle all the other way." He smiled at his father's
hands. "There was a bird, and the next morning that matters we married, and the
north will be born a man as well. Lord Tywin sinse your lord father's side."

"No." He was pleased to see the sort of squire to his feet. "It was the bride of
House Frey a whimpery, but I must ask them to let him stay a better life and a
sword and
```

Probamos una última vez el mismo input con una temperatura de 0.9.

[33]:
```
# Temperatura = 0.9
print(generate_text(model, start_string="Dragon ")) # generamos texto a partir␣
 ↪de la cadena "Dragon "
```

```
Dragon had been half-curious than he was, and there were almost penny, and his
wishes were their father on Pyke. Prousemon remains at Harrenhal. A knight at
the time he had the sort of maiden's son, nd made him feel too late. The weeds
reeled in the eyes of the e face of the highest table, but it was not like to be
properly as he would answer.

"My lord," said Melisandre with his bloody hand. "I did not come with you."

"The Tyrells were cold and nubbled at the chaos once Ser Horas Redwyne may be
pleas
```

Por último, probamos con el input "The" y las mismas temperaturas antes mencionadas.

[31]:
```
# Temperatura = 0.1
print(generate_text(model, start_string="The ")) # generamos texto a partir de␣
 ↪la cadena "The "
```

```
The will be the same as you can. The wildlings were true. The boy is the blood
of the dragon, she thought. I was told to see the shapes of the window and let
the fire had been so long as the star between them. The small brothers were so
far as the river was blowing on the stone walls of the corner of his head. A
slave was the one that might have been a strong sword in his hand. "I will not
say that a lie to you."

"I would have to say that the girl who was the one who lived the point of his
sworn sh
```

[27]:
```
# Temperatura = 0.5
print(generate_text(model, start_string="The ")) # generamos texto a partir de␣
 ↪la cadena "The "
```

```
The watched the next day, and a third so loud and short and sorry for that. "I
```

```
was soired a man he'd stay to the seven cry for his life. But the crown was
still, his bright black hair covered with blood and broken spears. The wind blew
on his back, twice, a bronze black beard, but a slope took him in the broken top
of his shore. He had a short time before the warmth of his golden spears and a
huge blade loose and then he could feel the steel on his legs. "I don't even
trust himself a septon proud, i
```

[34]:
```
# Temperatura = 0.9
print(generate_text(model, start_string="The ")) # generamos texto a partir de␣
 ↪la cadena "The "
```

```
The white sus or die all the world, though. We all are more armed more yanding
there are gone, don't talk for to see what you believe every one? Sansa knew the
man who died on her belly, then sounds well. It would not come to this masting
thousand men. Mady of the Iron Throne is the Lord of Light have laughed at one
of the day for a long most cruel way. Darry's the Fish kings had decreed dead it
down and form the Goat halls and fodder looked down at them with sweat. Jon sat
swamoped under his arm, w
```

1.2 Conclusiones

De acuerdo con los resultados obtenidos, podemos concluir que la temperatura en el contexto del modelado de lenguaje, especialmente en modelos de generación de texto como el que estás utilizando, es un parámetro clave que controla la aleatoriedad en la generación de texto. Básicamente, afecta la diversidad y la creatividad en los resultados generados.

- Una temperatura baja (0.1) produce resultados más deterministas, es decir, tiende a generar predicciones más seguras o probables basadas en el modelo. La salida es coherente y predecible, con repeticiones o patrones comunes.
- Una temperatura media (0.5) aumenta la aleatoriedad en comparación con 0.1, lo que conduce a una mayor variedad en la elección de palabras y estructuras gramaticales. Los resultados pueden ser más creativos y menos predecibles, pero aún mantienen cierta coherencia.
- Una temperatura alta (0.9) genera resultados altamente aleatorios y creativos. A menudo, produce texto más incoherente, con fragmentos que pueden carecer de sentido lógico, pero que aportan una gran dosis de creatividad. Las frases tienden a ser menos predecibles y coherentes.

En resumen, a medida que la temperatura aumenta, la aleatoriedad y la creatividad en los textos generados también aumentan, pero a expensas de la coherencia y la lógica en el texto. A temperaturas más bajas, los resultados tienden a ser más coherentes y predecibles, basados en patrones más sólidos aprendidos por el modelo en el corpus (el libro importado en un inicio). Por ello, es importante elegir la temperatura según el propósito de la generación de texto: coherencia versus creatividad.