莎士比亞機器人

# -\*- coding: utf-8 -\*-

"""text\_generation.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/zh-cn/tutorials/text/text\_generation.ipynb

##### Copyright 2019 The TensorFlow Authors.

"""

#@title Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# https://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""# 迴圈神經網路（RNN）文本生成

<table class="tfo-notebook-buttons" align="left">

<td>

<a target="\_blank" href="https://tensorflow.google.cn/tutorials/text/text\_generation"><img src="https://www.tensorflow.org/images/tf\_logo\_32px.png" />在 tensorflow.google.cn 上查看</a>

</td>

<td>

<a target="\_blank" href="https://colab.research.google.com/github/tensorflow/docs/blob/master/site/zh-cn/tutorials/text/text\_generation.ipynb"><img src="https://www.tensorflow.org/images/colab\_logo\_32px.png" />在 Google Colab 運行</a>

</td>

<td>

<a target="\_blank" href="https://github.com/tensorflow/docs/blob/master/site/zh-cn/tutorials/text/text\_generation.ipynb"><img src="https://www.tensorflow.org/images/GitHub-Mark-32px.png" />在 GitHub 上查看原始程式碼</a>

</td>

<td>

<a href="https://storage.googleapis.com/tensorflow\_docs/docs/site/zh-cn/tutorials/text/text\_generation.ipynb"><img src="https://www.tensorflow.org/images/download\_logo\_32px.png" />下載此 notebook</a>

</td>

</table>

本教程演示如何使用基於字元的 RNN 生成文本。我們將使用 Andrej Karpathy 在[《迴圈神經網路不合理的有效性》](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)一文中提供的莎士比亞作品資料集。給定此資料中的一個字元序列 （“Shakespear”），訓練一個模型以預測該序列的下一個字元（“e”）。通過重複調用該模型，可以生成更長的文本序列。

請注意：啟用 GPU 加速可以更快地執行此筆記本。在 Colab 中依次選擇：\*運行時 > 更改運行時類型 > 硬體加速器 > GPU\*。如果在本地運行，請確保 TensorFlow 的版本為 1.11 或更高。

本教程包含使用 [tf.keras](https://www.tensorflow.org/programmers\_guide/keras) 和 [eager execution](https://www.tensorflow.org/programmers\_guide/eager) 實現的可運行代碼。以下是當本教程中的模型訓練 30 個週期 （epoch），並以字串 “Q” 開頭時的示例輸出：

<pre>

QUEENE:

I had thought thou hadst a Roman; for the oracle,

Thus by All bids the man against the word,

Which are so weak of care, by old care done;

Your children were in your holy love,

And the precipitation through the bleeding throne.

BISHOP OF ELY:

Marry, and will, my lord, to weep in such a one were prettiest;

Yet now I was adopted heir

Of the world's lamentable day,

To watch the next way with his father with his face?

ESCALUS:

The cause why then we are all resolved more sons.

VOLUMNIA:

O, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, it is no sin it should be dead,

And love and pale as any will to that word.

QUEEN ELIZABETH:

But how long have I heard the soul for this world,

And show his hands of life be proved to stand.

PETRUCHIO:

I say he look'd on, if I must be content

To stay him from the fatal of our country's bliss.

His lordship pluck'd from this sentence then for prey,

And then let us twain, being the moon,

were she such a case as fills m

</pre>

雖然有些句子符合語法規則，但是大多數句子沒有意義。這個模型尚未學習到單詞的含義，但請考慮以下幾點：

\* 此模型是基於字元的。訓練開始時，模型不知道如何拼寫一個英文單詞，甚至不知道單詞是文本的一個單位。

\* 輸出文本的結構類似於劇本 -- 文字區塊通常以講話者的名字開始；而且與資料集類似，講話者的名字採用全大寫字母。

\* 如下文所示，此模型由小批次 （batch） 文本訓練而成（每批 100 個字元）。即便如此，此模型仍然能生成更長的文本序列，並且結構連貫。

## 設置

### 導入 TensorFlow 和其他庫

"""

# Commented out IPython magic to ensure Python compatibility.

from \_\_future\_\_ import absolute\_import, division, print\_function, unicode\_literals

try:

# %tensorflow\_version 僅存在於 Colab

# %tensorflow\_version 2.x

except Exception:

pass

import tensorflow as tf

import numpy as np

import os

import time

"""### 下載莎士比亞資料集

修改下面一行代碼，在你自己的資料上運行此代碼。

"""

path\_to\_file = tf.keras.utils.get\_file('shakespeare.txt', 'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')

"""### 讀取數據

首先，看一看文本：

"""

# 讀取並為 py2 compat 解碼

text = open(path\_to\_file, 'rb').read().decode(encoding='utf-8')

# 文本長度是指文本中的字元個數

print ('Length of text: {} characters'.format(len(text)))

# 看一看文本中的前 250 個字元

print(text[:250])

# 文本中的非重複字元

vocab = sorted(set(text))

print ('{} unique characters'.format(len(vocab)))

"""## 處理文本

### 向量化文本

在訓練之前，我們需要將字串映射到數位表示值。創建兩個查閱資料表格：一個將字元映射到數位，另一個將數位映射到字元。

"""

# 創建從非重複字元到索引的映射

char2idx = {u:i for i, u in enumerate(vocab)}

idx2char = np.array(vocab)

text\_as\_int = np.array([char2idx[c] for c in text])

"""現在，每個字元都有一個整數表示值。請注意，我們將字元映射至索引 0 至 `len(unique)`."""

print('{')

for char,\_ in zip(char2idx, range(20)):

print(' {:4s}: {:3d},'.format(repr(char), char2idx[char]))

print(' ...\n}')

# 顯示文本首 13 個字元的整數映射

print ('{} ---- characters mapped to int ---- > {}'.format(repr(text[:13]), text\_as\_int[:13]))

"""### 預測任務

給定一個字元或者一個字元序列，下一個最可能出現的字元是什麼？這就是我們訓練模型要執行的任務。輸入進模型的是一個字元序列，我們訓練這個模型來預測輸出 -- 每個時間步（time step）預測下一個字元是什麼。

由於 RNN 是根據前面看到的元素維持內部狀態，那麼，給定此時計算出的所有字元，下一個字元是什麼？

### 創建訓練樣本和目標

接下來，將文本劃分為樣本序列。每個輸入序列包含文本中的 `seq\_length` 個字元。

對於每個輸入序列，其對應的目標包含相同長度的文本，但是向右順移一個字元。

將文本拆分為長度為 `seq\_length+1` 的文字區塊。例如，假設 `seq\_length` 為 4 而且文本為 “Hello”， 那麼輸入序列將為 “Hell”，目標序列將為 “ello”。

為此，首先使用 `tf.data.Dataset.from\_tensor\_slices` 函數把文本向量轉換為字元索引流。

"""

# 設定每個輸入句子長度的最大值

seq\_length = 100

examples\_per\_epoch = len(text)//seq\_length

# 創建訓練樣本 / 目標

char\_dataset = tf.data.Dataset.from\_tensor\_slices(text\_as\_int)

for i in char\_dataset.take(5):

print(idx2char[i.numpy()])

"""`batch` 方法使我們能輕鬆把單個字元轉換為所需長度的序列。"""

sequences = char\_dataset.batch(seq\_length+1, drop\_remainder=True)

for item in sequences.take(5):

print(repr(''.join(idx2char[item.numpy()])))

"""對於每個序列，使用 `map` 方法先複製再順移，以創建輸入文本和目標文本。`map` 方法可以將一個簡單的函數應用到每一個批次 （batch）。"""

def split\_input\_target(chunk):

input\_text = chunk[:-1]

target\_text = chunk[1:]

return input\_text, target\_text

dataset = sequences.map(split\_input\_target)

"""列印第一批樣本的輸入與目標值："""

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()])))

"""這些向量的每個索引均作為一個時間步來處理。作為時間步 0 的輸入，模型接收到 “F” 的索引，並嘗試預測 “i” 的索引為下一個字元。在下一個時間步，模型執行相同的操作，但是 `RNN` 不僅考慮當前的輸入字元，還會考慮上一步的資訊。"""

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])))

"""### 創建訓練批次

前面我們使用 `tf.data` 將文本拆分為可管理的序列。但是在把這些資料輸送至模型之前，我們需要將資料重新排列 （shuffle） 並打包為批次。

"""

# 批大小

BATCH\_SIZE = 64

# 設定緩衝區大小，以重新排列資料集

# （TF 資料被設計為可以處理可能是無限的序列，

# 所以它不會試圖在記憶體中重新排列整個序列。相反，

# 它維持一個緩衝區，在緩衝區重新排列元素。）

BUFFER\_SIZE = 10000

dataset = dataset.shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE, drop\_remainder=True)

dataset

"""## 創建模型

使用 `tf.keras.Sequential` 定義模型。在這個簡單的例子中，我們使用了三個層來定義模型：

\* `tf.keras.layers.Embedding`：輸入層。一個可訓練的對照表，它會將每個字元的數位映射到一個 `embedding\_dim` 維度的向量。

\* `tf.keras.layers.GRU`：一種 RNN 類型，其大小由 `units=rnn\_units` 指定（這裡你也可以使用一個 LSTM 層）。

\* `tf.keras.layers.Dense`：輸出層，帶有 `vocab\_size` 個輸出。

"""

# 詞集的長度

vocab\_size = len(vocab)

# 嵌入的維度

embedding\_dim = 256

# RNN 的單元數量

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)

"""對於每個字元，模型會查找嵌入，把嵌入當作輸入運行 GRU 一個時間步，並用密集層生成邏輯回歸 （logits），預測下一個字元的對數可能性。

![資料在模型中傳輸的示意圖](https://github.com/littlebeanbean7/docs/blob/master/site/en/tutorials/text/images/text\_generation\_training.png?raw=1)

## 試試這個模型

現在運行這個模型，看看它是否按預期運行。

首先檢查輸出的形狀：

"""

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)")

"""在上面的例子中，輸入的序列長度為 `100`， 但是這個模型可以在任何長度的輸入上運行："""

model.summary()

"""為了獲得模型的實際預測，我們需要從輸出分佈中抽樣，以獲得實際的字元索引。這個分佈是根據對字元集的邏輯回歸定義的。

請注意：從這個分佈中 \_抽樣\_ 很重要，因為取分佈的 \_最大值引數點集（argmax）\_ 很容易使模型卡在迴圈中。

試試這個批次中的第一個樣本：

"""

sampled\_indices = tf.random.categorical(example\_batch\_predictions[0], num\_samples=1)

sampled\_indices = tf.squeeze(sampled\_indices,axis=-1).numpy()

"""這使我們得到每個時間步預測的下一個字元的索引。"""

sampled\_indices

"""解碼它們，以查看此未經訓練的模型預測的文本："""

print("Input: \n", repr("".join(idx2char[input\_example\_batch[0]])))

print()

print("Next Char Predictions: \n", repr("".join(idx2char[sampled\_indices ])))

"""## 訓練模型

此時，這個問題可以被視為一個標準的分類問題：給定先前的 RNN 狀態和這一時間步的輸入，預測下一個字元的類別。

### 添加優化器和損失函數

標準的 `tf.keras.losses.sparse\_categorical\_crossentropy` 損失函數在這裡適用，因為它被應用於預測的最後一個維度。

因為我們的模型返回邏輯回歸，所以我們需要設定命令列參數 `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())

"""使用 `tf.keras.Model.compile` 方法配置訓練步驟。我們將使用 `tf.keras.optimizers.Adam` 並採用預設參數，以及損失函數。"""

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

"""### 配置檢查點

使用 `tf.keras.callbacks.ModelCheckpoint` 來確保訓練過程中保存檢查點。

"""

# 檢查點保存至的目錄

checkpoint\_dir = './training\_checkpoints'

# 檢查點的檔案名

checkpoint\_prefix = os.path.join(checkpoint\_dir, "ckpt\_{epoch}")

checkpoint\_callback=tf.keras.callbacks.ModelCheckpoint(

filepath=checkpoint\_prefix,

save\_weights\_only=True)

"""### 執行訓練

為保持訓練時間合理，使用 10 個週期來訓練模型。在 Colab 中，將運行時設置為 GPU 以加速訓練。

"""

EPOCHS=10

history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint\_callback])

"""## 生成文本

### 恢復最新的檢查點

為保持此次預測步驟簡單，將批大小設定為 1。

由於 RNN 狀態從時間步傳遞到時間步的方式，模型建立好之後只接受固定的批大小。

若要使用不同的 `batch\_size` 來運行模型，我們需要重建模型並從檢查點中恢復權重。

"""

tf.train.latest\_checkpoint(checkpoint\_dir)

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()

"""### 預測迴圈

下面的代碼塊生成文本：

\* 首先設置起始字串，初始化 RNN 狀態並設置要生成的字元個數。

\* 用起始字串和 RNN 狀態，獲取下一個字元的預測分佈。

\* 然後，用分類分佈計算預測字元的索引。把這個預測字元當作模型的下一個輸入。

\* 模型返回的 RNN 狀態被輸送回模型。現在，模型有更多上下文可以學習，而非只有一個字元。在預測出下一個字元後，更改過的 RNN 狀態被再次輸送回模型。模型就是這樣，通過不斷從前面預測的字元獲得更多上下文，進行學習。

![為生成文本，模型的輸出被輸送回模型作為輸入](https://github.com/littlebeanbean7/docs/blob/master/site/en/tutorials/text/images/text\_generation\_sampling.png?raw=1)

查看生成的文本，你會發現這個模型知道什麼時候使用大寫字母，什麼時候分段，而且模仿出了莎士比亞式的詞彙。由於訓練的週期小，模型尚未學會生成連貫的句子。

"""

def generate\_text(model, start\_string):

# 評估步驟（用學習過的模型生成文本）

# 要生成的字元個數

num\_generate = 1000

# 將起始字串轉換為數位（向量化）

input\_eval = [char2idx[s] for s in start\_string]

input\_eval = tf.expand\_dims(input\_eval, 0)

# 空字串用於存儲結果

text\_generated = []

# 低溫度會生成更可預測的文本

# 較高溫度會生成更令人驚訝的文本

# 可以通過試驗以找到最好的設定

temperature = 1.0

# 這裡批大小為 1

model.reset\_states()

for i in range(num\_generate):

predictions = model(input\_eval)

# 刪除批次的維度

predictions = tf.squeeze(predictions, 0)

# 用分類分佈預測模型返回的字元

predictions = predictions / temperature

predicted\_id = tf.random.categorical(predictions, num\_samples=1)[-1,0].numpy()

# 把預測字元和前面的隱藏狀態一起傳遞給模型作為下一個輸入

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: "))

"""若想改進結果，最簡單的方式是延長訓練時間 （試試 `EPOCHS=30`）。

你還可以試驗使用不同的起始字串，或者嘗試增加另一個 RNN 層以提高模型的準確率，亦或調整溫度參數以生成更多或者更少的隨機預測。

## 高級：自訂訓練

上面的訓練步驟簡單，但是能控制的地方不多。

至此，你已經知道如何手動運行模型。現在，讓我們打開訓練迴圈，並自己實現它。這是一些任務的起點，例如實現 \_課程學習\_ 以説明穩定模型的開環輸出。

你將使用 `tf.GradientTape` 跟蹤梯度。關於此方法的更多資訊請參閱 [eager execution 指南](https://www.tensorflow.org/guide/eager)。

步驟如下：

\* 首先，初始化 RNN 狀態，使用 `tf.keras.Model.reset\_states` 方法。

\* 然後，反覆運算資料集（逐批次）並計算每次反覆運算對應的 \*預測\*。

\* 打開一個 `tf.GradientTape` 並計算該上下文時的預測和損失。

\* 使用 `tf.GradientTape.grads` 方法，計算當前模型變數情況下的損失梯度。

\* 最後，使用優化器的 `tf.train.Optimizer.apply\_gradients` 方法向下邁出一步。

"""

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

# 訓練步驟

EPOCHS = 10

for epoch in range(EPOCHS):

start = time.time()

# 在每個訓練週期開始時，初始化隱藏狀態

# 隱藏狀態最初為 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))

# 每 5 個訓練週期，保存（檢查點）1 次模型

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))