PART 2 TensorFlow

6. Workshop 4 : 自然語言處理 (NLP)

< NLP > : Text generation with LSTM - `Generative Deep Learning`

> **[Reference] :**
>
> - FRANÇOIS CHOLLET, **Deep Learning with Python**, Chapter 8, Section 1, Manning, 2018. (https://tanthiamhuat.files.wordpress.com/2018/03/deeplearningwithpython.pdf (https://tanthiamhuat.files.wordpress.com/2018/03/deeplearningwithpython.pdf))

In [1]:

```
import keras
keras.__version__
```

```
/Users/macmini1/anaconda3/lib/python3.6/site-packages/h5p
y/__init__.py:36: FutureWarning: Conversion of the second
argument of issubdtype from `float` to `np.floating` is de
precated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_conv
erters
Using TensorFlow backend.
```

Out[1]:

```
'2.2.4'
```

Implementing character-level LSTM text generation

- Let's put these ideas in practice in a Keras implementation. The first thing we need is a lot of text data that we can use to learn a language model.
- You could use any sufficiently large text file or set of text files -- Wikipedia, the Lord of the Rings, etc.

> - In this example we will use some of the writings of Nietzsche, the late-19th century German philosopher (translated to English).
> - The language model we will learn will thus be specifically a model of Nietzsche's writing style and topics of choice, rather than a more generic model of the English language.

- **The following diagram is from the book - `Deep Learning with Python` , Chapter 8, Section 8.1.2.**

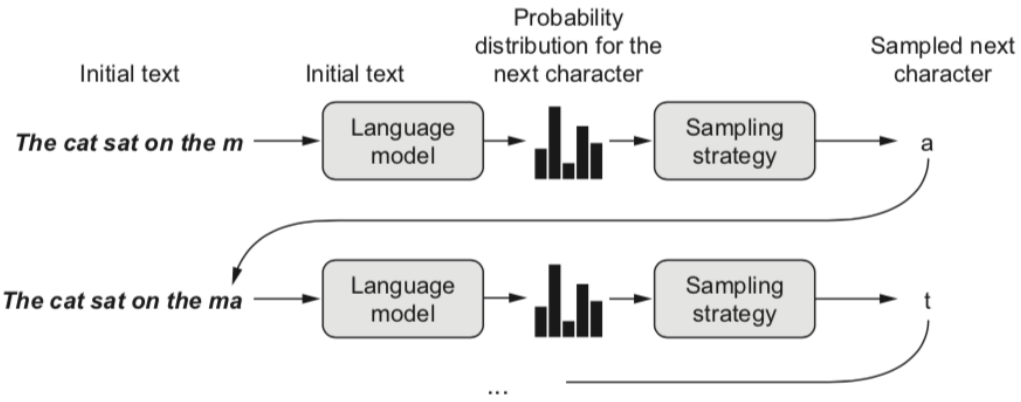

Figure 8.1 The process of character-by-character text generation using a language model

Preparing the data

- **Let's start by downloading the corpus and converting it to lowercase:**

In [2]:

```
import numpy as np

path = keras.utils.get_file(
    'nietzsche.txt',
    origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')
text = open(path).read().lower()
print('Corpus length:', len(text))  ## Corpus length (= len(text)):
```

```
Corpus length: 600893
```

- **Next, we will extract partially-overlapping sequences of length `maxlen` , one-hot encode them and pack them in a 3D Numpy array `x` of shape `(sequences, maxlen, unique_characters)` .**

In [3]:

```
# maxlen : Length of extracted character sequences
maxlen = 60   ## You'll extract sequences of 60 characters.

# step : sampling a new sequence every `step` characters
step = 3      ## You'll sample a new sequence every three character

# This holds our extracted sequences
sentences = []   ## Holding the extracted sequences...

# This holds the targets (the follow-up characters)
next_chars = []

for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('Number of sequences:', len(sentences))  ##  len(sentences) =
```

```
Number of sequences: 200278
```

In [4]:

```
text[0:maxlen]  ## sentences[0]
```

Out[4]:

```
'preface\n\n\nsupposing that truth is a woman--what then?
is the'
```

In [5]:

```
text[0 + maxlen]  ## next_chars[0]
```

Out[5]:

```
'r'
```

In [6]:

```
text[3 : 3+maxlen]  ## sentences[1]
```

Out[6]:

```
'face\n\n\nsupposing that truth is a woman--what then? is
there '
```

In [7]:

```
text[3 + maxlen]  ## next_chars[1]
```

Out[7]:

```
'n'
```

In [8]:

```
text[:1000]
```

Out[8]:

```
'preface\n\n\nsupposing that truth is a woman--what then?
is there not ground\nfor suspecting that all philosophers,
in so far as they have been\ndogmatists, have failed to un
derstand women--that the terrible\nseriousness and clumsy
importunity with which they have usually paid\ntheir addre
sses to truth, have been unskilled and unseemly methods fo
r\nwinning a woman? certainly she has never allowed hersel
f to be won; and\nat present every kind of dogma stands wi
th sad and discouraged mien--if,\nindeed, it stands at al
l! for there are scoffers who maintain that it\nhas falle
n, that all dogma lies on the ground--nay more, that it is
at\nits last gasp. but to speak seriously, there are good
grounds for hoping\nthat all dogmatizing in philosophy, wh
atever solemn, whatever conclusive\nand decided airs it ha
s assumed, may have been only a noble puerilism\nand tyron
ism; and probably the time is at hand when it will be once
\nand again understood what has actually sufficed for the
basis of such\nimposing and abso'
```

In [9]:

```
# List of unique characters in the corpus
chars = sorted(list(set(text)))
print('Unique characters:', len(chars))

# Dictionary mapping unique characters to their index in `chars`
char_indices = dict((char, chars.index(char)) for char in chars)
```

```
Unique characters: 57
```

- **Simultaneously, we prepare a array `y` containing the corresponding targets: the one-hot encoded characters that come right after each extracted sequence.**

In [10]:

```
# Next, one-hot encode the characters into binary arrays.
print('Vectorization...')
x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
```

```
Vectorization...
```

Building the network with `Keras/TensorFlow`

- **Our network is a single `LSTM` layer followed by a `Dense` classifier and softmax over all possible characters.**

[NOTE] : Recurrent neural networks are not the only way to do sequence data generation; `1D convnets` also have proven extremely successful at it in recent times.

In [11]:

```
from keras import layers

model = keras.models.Sequential()
model.add(layers.LSTM(128, input_shape=(maxlen, len(chars))))
model.add(layers.Dense(len(chars), activation='softmax'))
```

Since our targets are one-hot encoded, we will use `categorical_crossentropy` as the loss to train the model:

In [12]:

```
optimizer = keras.optimizers.RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

Training the language model and sampling from it

The importance of the sampling strategy :

> - A more interesting approach makes slightly more sur- prising choices: it introduces `randomness` in the sampling process, by sampling from the probability distribution for the next character. This is called `stochastic sampling` (recall that *stochasticity* is what we call *randomness* in this field).
> - In order to control the amount of stochasticity in the sampling process, we'll intro- duce a parameter called the `softmax temperature` that characterizes the entropy of the probability distribution used for sampling: it characterizes how surprising or predict- able the choice of the next character will be. *Given a temperature value, a new probability distribution is computed from the original one* (the softmax output of the model) by reweighting it in the following way.

Given a trained model and a seed text snippet, we generate new text by repeatedly:

- Drawing from the model a probability distribution over the next character given the text available so far
- Reweighting the distribution to a certain `temperature`
- Sampling the next character at random according to the reweighted distribution
- Adding the new character at the end of the available text

[NOTE] : `temperature` is a factor quantifying the entropy of the output distribution.

This is the code we use to `reweight the original probability distribution` coming out of the model, and draw a character index from it (the "sampling function"):

In [13]:

```
##  Original_distribution is a 1D Numpy array of probability values
##  'temperature' is a factor quantifying the entropy of the output
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    ##------------------------------------------------------------
    ##  Computeing a reweighted version of the original distributio
    ##  The sum of the distribution may no longer be 1, so you divi
    ##  by its sum to obtain the new distribution.
    ##------------------------------------------------------------
    preds = exp_preds / np.sum(exp_preds)

    ##-------------------------------------------------------
    ##  Draw samples from a multinomial distribution.
    ##  The multinomial distribution is a multivariate
    ##  generalisation of the binomial distribution.
    ##-------------------------------------------------------
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

Finally, this is the loop where we repeatedly train and generated text.

- **We start generating text using a range of different temperatures after every epoch.**
- **This allows us to see how the generated text evolves as the model starts converging, as well as the impact of temperature in the sampling strategy.**

In [15]:

```
import random
import sys

for epoch in range(1, 4):  ## Trains the model for 60 epochs
    print('epoch', epoch)
    # Fit the model for 1 epoch on the available training data
    model.fit(x, y,
              batch_size=128,
              epochs=1)

    # Select a text seed at random
    start_index = random.randint(0, len(text) - maxlen - 1)
    generated_text = text[start_index: start_index + maxlen]
    print('--- Generating with seed: "' + generated_text + '"')

    for temperature in [0.2, 0.5, 1.0, 1.2]:
        print('------ temperature:', temperature)
        sys.stdout.write(generated_text)

        # We generate 400 characters
        for i in range(400):
            sampled = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(generated_text):
                sampled[0, t, char_indices[char]] = 1.

            preds = model.predict(sampled, verbose=0)[0]
            next_index = sample(preds, temperature)
            next_char = chars[next_index]

            generated_text += next_char
            generated_text = generated_text[1:]

            sys.stdout.write(next_char)
            sys.stdout.flush()
        print()
```

```
mofiers, dmposing.

shoochers
ror--
befevife
youn,
for we lass sumposowher condeesscisions, anowlered--acco
mden thai hir "wilt, worthexs, it would pol
muctly opile. hes berivadi
epoch 2
Epoch 1/1
200278/200278 [==============================] - 238s 1m
s/step - loss: 1.6314
--- Generating with seed: "timate product of the fear of
truth, as artist-adoration
and"
------ temperature: 0.2
timate product of the fear of truth, as artist-adoration
and the most decised and the something the sense of the
sense of the sense of the sense of the souls and the som
```

At epoch 60, the model has mostly converged, and the text starts to look significantly more coherent.

Here's the result with temperature = 0.2 :

```
 cheerfulness, friendliness and kindness of a heart are the
sense of the
spirit is a man with the sense of the sense of the world
of the
self-end and self-concerning the subjection of the
strengthorixes--the
subjection of the subjection of the subjection of the
self-concerning the feelings in the superiority in the
subjection of the
subjection of the spirit isn't to be a man of the sense of
the
subjection and said to the strength of the sense of the
```

Here's the result with temperature = 0.5 :

```
 cheerfulness, friendliness and kindness of a heart are the
part of the soul
who have been the art of the philosophers, and which the
one
won't say, which is it the higher the and with religion of
the frences.
the life of the spirit among the most continuess of the
strengther of the sense the conscience of men of precisely
before enough
presumption, and can mankind, and something the
conceptions, the
subjection of the sense and suffering and the
```

Here's the result with temperature = 1.0 :

```
 cheerfulness, friendliness and kindness of a heart are
spiritual by the
ciuture for the
entalled is, he astraged, or errors to our you idstood--
and it needs,
to think by spars to whole the amvives of the newoatly,
prefectly
raals! it was
name, for example but voludd atu-especity"--or rank onee,
or even all
"solett increessic of the world and
implussional tragedy experience, transf, or insiderar,--
must hast
if desires of the strubction is be stronges
```

[Analysis] :

- **As you can see, a low temperature value results in extremely repetitive and predictable text, but local structure is highly realistic: in particular, all words (a word being a local pattern of characters) are real English words.**
- **With higher temperatures, the generated text becomes more interesting, surprising, even creative; it sometimes invents completely new words that sound somewhat plausible (such as eterned and troveration). With a high temperature, the local structure starts to break down, and most words look like semi-random strings of characters.**
- Without a doubt, 0.5 is the most interesting temperature for text generation in this specific setup. Always experiment with multiple sampling strategies! A clever balance between learned structure and random- ness is what makes generation interesting.
- Note that by training a bigger model, longer, on more data, you can achieve generated samples that look much more coherent and realistic than this one.
- But, of course, don't expect to ever generate any meaningful text, other than by random chance: all you're doing is sampling data from a statistical model of which characters come after which characters.
- **Language is a communication channel, and there's a distinction between what communications are about and the statistical structure of the messages in which communications are encoded.**

< Exercise > :

請將上述案例程式 (Keras/TensorFlow)，改寫成 TensorFlow 程式，並輸出結果。