In [1]:

```
from __future__ import absolute_import, division, print_function, unicode_literals

!pip install -q tensorflow-gpu==2.0.0
!pip install -q tensorflow==2.0.0
import tensorflow as tf

import numpy as np
import os
import time
```

In [2]:

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

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/
shakespeare.txt
1122304/1115394 [==============================] - 0s 0us/step
```

In [3]:

```
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')

print('텍스트의 길이: {}자'.format(len(text)))
```

```
텍스트의 길이: 1115394자
```

In [4]:

```
print(text[:250])
```

```
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.
```

In [5]:

```
vocab = sorted(set(text))

print('고유 문자수 {}개'.format(len(vocab)))
```

```
고유 문자수 65개
```

In [6]:

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

In [8]:

```
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,
  ...
}
```

In [9]:

```
print ('{} ---- 문자들이 다음의 정수로 매핑되었습니다 ----> {}'.format(repr(text[:13]), text_as_int[:13]))
```

```
'First Citizen' ---- 문자들이 다음의 정수로 매핑되었습니다 ----> [18 47 56 57 58
 1 15 47 58 47 64 43 52]
```

In [10]:

```
seq_length = 100
examples_per_epoch = len(text)

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
```

In [11]:

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

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

```
'First Citizen:WnBefore we proceed any further, hear me speak.WnWnAll:WnSpeak, spe
ak.WnWnFirst Citizen:WnYou '
'are all resolved rather to die than to famish?WnWnAll:WnResolved. resolved.WnWnFi
rst Citizen:WnFirst, you k'
"now Caius Marcius is chief enemy to the people.WnWnAll:WnWe know't, we know't.Wn
WnFirst Citizen:WnLet us ki"
"ll him, and we'll have corn at our own price.WnIs't a verdict?WnWnAll:WnNo more t
alking on't; let it be d"
'one: away, away!WnWnSecond Citizen:WnOne word, good citizens.WnWnFirst Citizen:Wn
We are accounted poor citi'
```

In [12]:

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

dataset = sequences.map(split_input_target)
```

In [13]:

```
for input_example, target_example in dataset.take(1):
    print('입력 데이터: ', repr(''.join(idx2char[input_example.numpy()])))
    print('타깃 데이터: ', repr(''.join(idx2char[target_example.numpy()])))
```

```
입력 데이터:  'First Citizen:WnBefore we proceed any further, hear me speak.WnWnAl
l:WnSpeak, speak.WnWnFirst Citizen:WnYou'
타깃 데이터:  'irst Citizen:WnBefore we proceed any further, hear me speak.WnWnAl
l:WnSpeak, speak.WnWnFirst Citizen:WnYou '
```

In [14]:

```
for i, (input_idx, target_idx) in enumerate(zip(input_example[:5], W
                                                target_example[:5])):
    print("{:4d}단계".format(i))
    print("  입력: {} ({:s})".format(input_idx, repr(idx2char[input_idx])))
    print("  예상 출력: {} ({:s})".format(target_idx, repr(idx2char[target_idx])))
```

```
   0단계
  입력: 18 ('F')
  예상 출력: 47 ('i')
   1단계
  입력: 47 ('i')
  예상 출력: 56 ('r')
   2단계
  입력: 56 ('r')
  예상 출력: 57 ('s')
   3단계
  입력: 57 ('s')
  예상 출력: 58 ('t')
   4단계
  입력: 58 ('t')
  예상 출력: 1 (' ')
```

In [15]:

```
BATCH_SIZE = 64
BUFFER_SIZE = 10000

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

dataset
```

Out[15]:

```
<BatchDataset shapes: ((64, 100), (64, 100)), types: (tf.int32, tf.int32)>
```

In [39]:

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

In [40]:

```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embeddig_dim, batch_input_shape = W
                                  [batch_size, None]),
        tf.keras.layers.LSTM(rnn_units, return_sequences = True, W
                                    recurrent_initializer = 'glorot_uniform'),
        tf.keras.layers.Dense(vocab_size)
    ])
    return model
```

In [41]:

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

In [21]:

```
for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "# (배치 크기, 시퀀스 길이, 어휘 사전 크기)")
```

(64, 100, 65) # (배치 크기, 시퀀스 길이, 어휘 사전 크기)

In [23]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(64, None, 256)	16640
lstm (LSTM)	(64, None, 1024)	5246976
dense (Dense)	(64, None, 65)	66625

Total params: 5,330,241
Trainable params: 5,330,241
Non-trainable params: 0

In [25]:

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

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

In [26]:

```
sampled_indices
```

Out[26]:

```
array([ 4, 54, 41, 49, 53, 47, 32,  8,  9, 59, 36, 32, 10, 15, 20, 14, 11,
       35, 50, 58, 14, 34,  4, 63, 62, 17, 59, 36, 50, 44, 37, 13, 25, 64,
       53, 39, 20,  6, 25, 64,  1,  4, 56, 56, 41, 40, 37, 49, 19, 20, 54,
       64, 42, 36, 57, 34, 33, 58,  0, 46, 47, 59, 54,  7, 39, 24, 49, 56,
       27, 39, 64, 46, 30, 21, 54, 45, 41, 53,  6, 23, 20, 44, 57, 14, 21,
       24, 15, 31, 40, 60, 24, 25, 50, 50, 23, 31,  9, 38, 63, 55],
      dtype=int64)
```

In [27]:

```
print("입력: Wn", repr("".join(idx2char[input_example_batch[0]])))
print()
print("예측된 다음 문자: Wn", repr("".join(idx2char[sampled_indices ])))
```

입력:
 'hen the lion fawns upon the lamb,WnThe lamb will never cease to follow him.WnWnE
XETER:WnHark, hark, my l'

예측된 다음 문자:
 '&pckoiT.3uXT:CHB:WltBV&yxEuXlfYAMzoaH,Mz &rrcbYkGHpzdXsVUtWnhiup-aLkr0azhRlpgco,
KHfsBILCSbvLMlIKS3Zyq'

In [28]:

```
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("예측 배열 크기(shape): ", example_batch_predictions.shape,
      " # (배치 크기, 시퀀스 길이, 어휘 사전 크기")
print("스칼라 손실:       ", example_batch_loss.numpy().mean())
```

예측 배열 크기(shape): (64, 100, 65) # (배치 크기, 시퀀스 길이, 어휘 사전 크기
스칼라 손실: 4.174767

In [29]:

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

In [30]:

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

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

In [31]:

```
EPOCHS = 10
```

In [32]:

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

```
Epoch 1/10
172/172 [==============================] - 11s 66ms/step - loss: 2.6787
Epoch 2/10
172/172 [==============================] - 9s 49ms/step - loss: 1.9797
Epoch 3/10
172/172 [==============================] - 8s 49ms/step - loss: 1.7103
Epoch 4/10
172/172 [==============================] - 8s 48ms/step - loss: 1.5540
Epoch 5/10
172/172 [==============================] - 9s 50ms/step - loss: 1.4570
Epoch 6/10
172/172 [==============================] - 9s 50ms/step - loss: 1.3917
Epoch 7/10
172/172 [==============================] - 9s 52ms/step - loss: 1.3401
Epoch 8/10
172/172 [==============================] - 9s 52ms/step - loss: 1.2978
Epoch 9/10
172/172 [==============================] - 9s 50ms/step - loss: 1.2609
Epoch 10/10
172/172 [==============================] - 9s 51ms/step - loss: 1.2269
```

In [33]:

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

Out[33]:

```
'./training_checkpointsWWckpt_10'
```

In [42]:

```
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)
model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
model.build(tf.TensorShape([1, None]))
```

In [43]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (1, None, 256)            16640
_________________________________________________________________
lstm_2 (LSTM)                (1, None, 1024)           5246976
_________________________________________________________________
dense_2 (Dense)              (1, None, 65)             66625
=================================================================
Total params: 5,330,241
Trainable params: 5,330,241
Non-trainable params: 0
_________________________________________________________________
```

In [48]:

```
def generate_text(model, start_string):
    num_generate = 1000

    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []
    temperature = 1.0

    model.reset_states()
    for i in range(num_generate):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)
        predictions = predictions / temperature

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

        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(idx2char[predicted_id])
    return (start_string + ''.join(text_generated))
```

In [49]:

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

```
ROMEO: G jenthy hous, thounelell furs ucht he' a l pre chaRUvet,


Nl ban's fofes l-, t
Hosote mer f;
TIlt ongursthave.
ANCis theven iupr t wen'sthequg thetwofa thin l lmyod Y wined e k fo, t ar by lsth
ineme? he l h, eser l th y, by.
Wher weseme? yorore fave t.
thokin angoore:
ENBUSABitere' he HAND t benit ul LOLOUConeeryoong.
Y pt, then's angnd g-dugnherongith ceve ankis buse? sontnk aturero hed batous che
d!
S:

t, me ted,

SblyowOLemuls ou to.
Prser?
By?
HA nthour sthar:

LILl ful o hit lerewoves thend dakis w?

Whias 'd was mobul MlO:
P; he pe bevansire d, w y ciger
Hl'sthe am angseancubye'st theathif pr'sthan?
T$ m mant, bathisinod hiscowhifowl whre Geneare f gouece yo serend ther man;
lito-k w

LENais oue y, hinofane towert t, ang te ourowap?
Ellan yo'l acroraMy breas d and tord bupershat wexe theyous icoutee d lEKee m se s
f tsties horLE:

ADr$meraringose dinte l siveoower dourend ars fus;
Wis s aze?
S: y,
O?
isome l NRere AREWAnqut s mangead weNond gngen g hores
Timeane'VO; y
O:
GRAnoo
```

In [50]:

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

In [51]:

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

In [52]:

```
@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
```

In [53]:

```
EPOCHS = 10

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

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

        if batch_n % 100 == 0:
            template = '에포크 {} 배치 {} 손실 {}'
            print(template.format(epoch+1, batch_n, loss))

    if (epoch + 1) % 5 == 0:
        model.save_weights(checkpoint_prefix.format(epoch=epoch))

    print ('에포크 {} 손실 {:.4f}'.format(epoch+1, loss))
    print ('1 에포크 당 {}초 소요Wn'.format(time.time() - start))

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

```
에포크 1 배치 0 손실 4.175161838531494
에포크 1 배치 100 손실 2.436166286468506
에포크 1 손실 2.1985
1 에포크 당 9.77225923538208초 소요

에포크 2 배치 0 손실 2.2057385444641113
에포크 2 배치 100 손실 1.9506747722625732
에포크 2 손실 1.8633
1 에포크 당 7.557368516921997초 소요

에포크 3 배치 0 손실 1.8185088634490967
에포크 3 배치 100 손실 1.7232131958007812
에포크 3 손실 1.6270
1 에포크 당 7.671477794647217초 소요

에포크 4 배치 0 손실 1.6551377773284912
에포크 4 배치 100 손실 1.5309635400772095
에포크 4 손실 1.5450
1 에포크 당 7.673476457595825초 소요

에포크 5 배치 0 손실 1.4839167594909668
에포크 5 배치 100 손실 1.461898684501648
에포크 5 손실 1.4642
1 에포크 당 8.00175404548645초 소요

에포크 6 배치 0 손실 1.431877851486206
에포크 6 배치 100 손실 1.3933476209640503
에포크 6 손실 1.3746
1 에포크 당 7.698498964309692초 소요

에포크 7 배치 0 손실 1.4131022691726685
에포크 7 배치 100 손실 1.3328430652618408
에포크 7 손실 1.3542
1 에포크 당 7.58240008354187초 소요

에포크 8 배치 0 손실 1.3257560729980469
에포크 8 배치 100 손실 1.320218801498413
에포크 8 손실 1.3344
1 에포크 당 7.689500570297241초 소요

에포크 9 배치 0 손실 1.3344112634658813
에포크 9 배치 100 손실 1.2855677604675293
에포크 9 손실 1.2917
1 에포크 당 7.681474447250366초 소요

에포크 10 배치 0 손실 1.273826003074646
에포크 10 배치 100 손실 1.28106689453125
에포크 10 손실 1.2839
1 에포크 당 7.849625110626221초 소요
```

In [ ]: