Text Generation With LSTM RNN in Python with Keras

Supervisor: Dr.Abdul Qayyum

Team
- Vasileios Melissianos
- Ahmed Hossameldin
- Qingqing Nie

UNIVERSITÉ DE BOURGOGNE

TABLE OF CONTENTS

01 Introduction

- RNN
- LSTM

02 Project overview

- Project introduction
- Dataset Preparation

03 LSTM RNN Training Model

- Training Model architecture

04 LSTM RNN Generating text Model

- Generating Model architecture

05 Results and Discussion

- Results
- Conclusions
- References

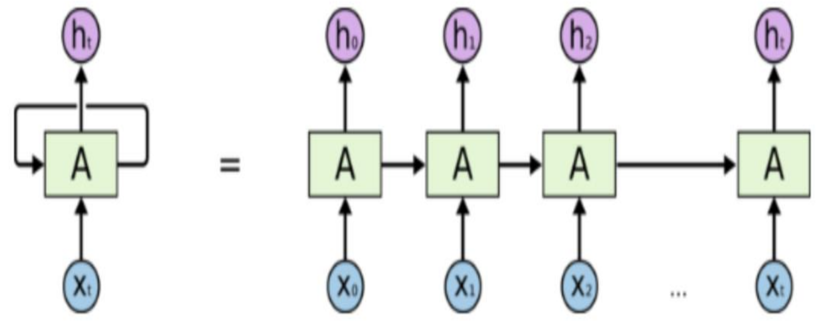
01

Introduction

RNN

What is Recurrent Neural Network (RNN)?

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

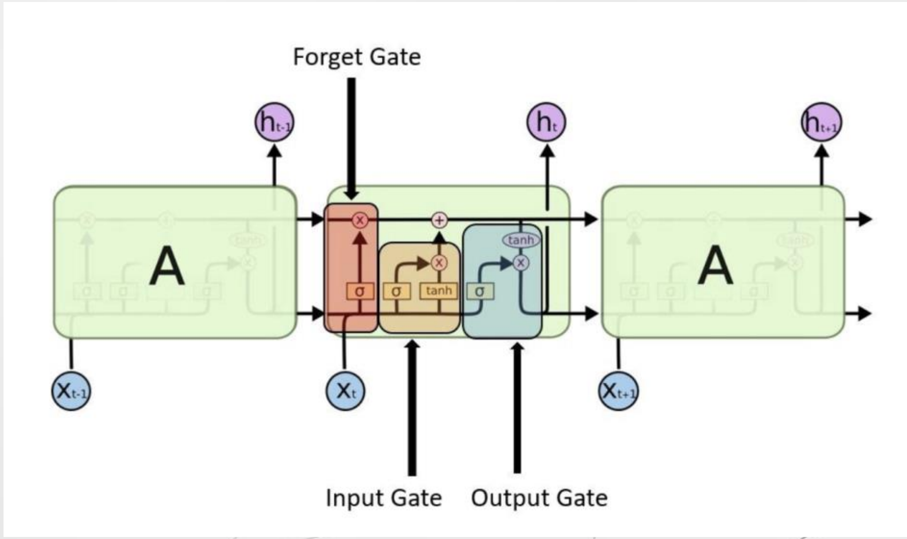


An unrolled recurrent neural network.

LSTM

What is Long Short Term Memory (LSTM)?

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:



02

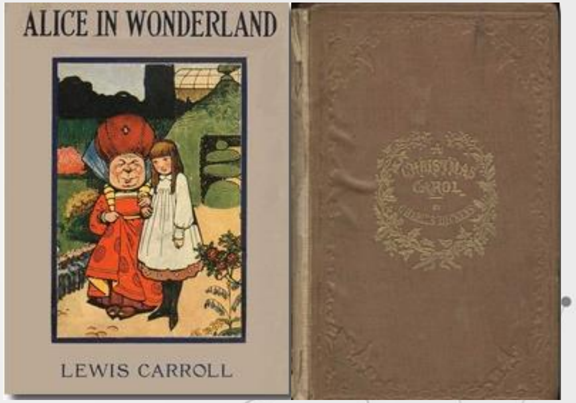
Project overview

Project introduction

we may also experiment with other ASCII data, such as computer source code, marked documents in LaTeX, HTML or Markdown, and more. These experiments are not limited to text.

- One of the usage of Recurrent neural networks is to generate models by learning sequences of problem and then generate reasonable sequences for this problem.
- Generative models used to know how good the models have learned about the problem and to know more about the problem.
- Our project is to create a generative model for character-by-character text using LSTM RNN in Python with Keras packages.

Dataset Preparation



- We will use two open source famous text books as our datasets from Project Gutenberg:

1-Alice's Adventures in Wonderland by Lewis Carroll.
2-A Christmas Carol in Prose; Being a Ghost Story of Christmas by Charles Dickens.

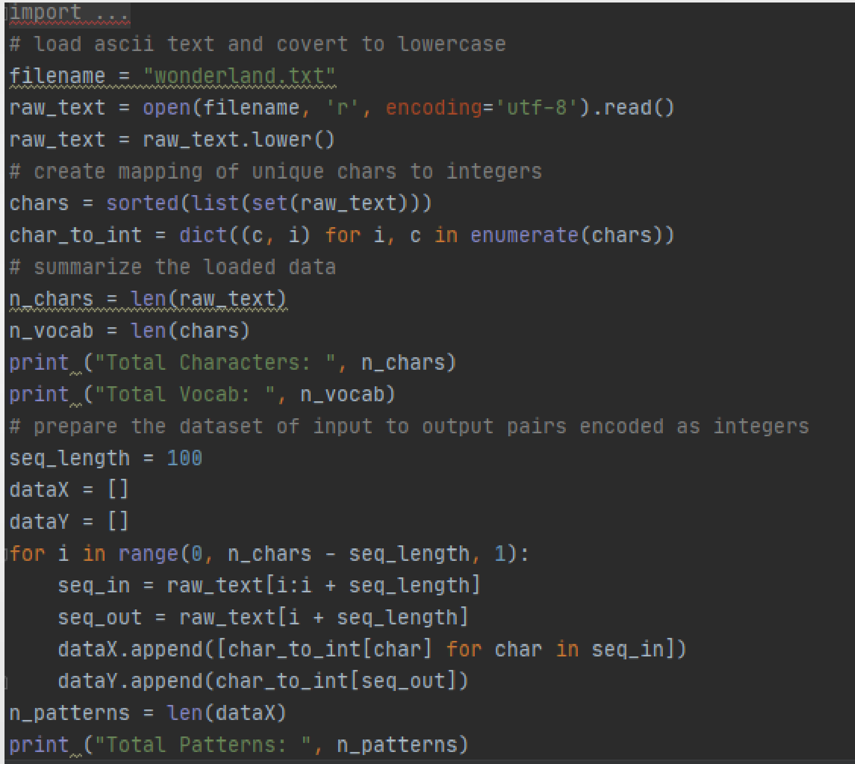
- We will copy the ASCII format (Plain Text UTF-8) of the two books starting from (**starting of…) to (The end) without these two lines and save them in two text files with the names of “wonderland.txt” and “Christmas-Carol” in the same folder of the code.

03

LSTM RNN Training Model

Training Model architecture

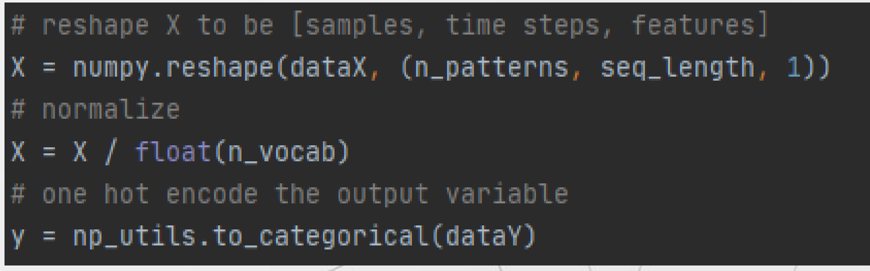
- Firstly we created an Anaconda environment and install numpy, Keras and tensorflow packages.
- Then we wrote our code that create a LSTM RNN model.
- The sequence of the code is first, load our ASCII text and convert it to lowercase characters.
- Then we have to prepare our dataset by converting it from characters to integers. That will be by creating a unique integers map of all of the distinct characters and print the number of total characters and total number of the distinct characters (the number of distinct characters of the first model is 43).
- The next step is to define the training data. In our code we will split our dataset into a sequence of 100 randomly.



```python
import ...
# load ascii text and covert to lowercase
filename = "wonderland.txt"
raw_text = open(filename, 'r', encoding='utf-8').read()
raw_text = raw_text.lower()
# create mapping of unique chars to integers
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
# summarize the loaded data
n_chars = len(raw_text)
n_vocab = len(chars)
print ("Total Characters: ", n_chars)
print ("Total Vocab: ", n_vocab)
# prepare the dataset of input to output pairs encoded as integers
seq_length = 100
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
    seq_in = raw_text[i:i + seq_length]
    seq_out = raw_text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print ("Total Patterns: ", n_patterns)
```

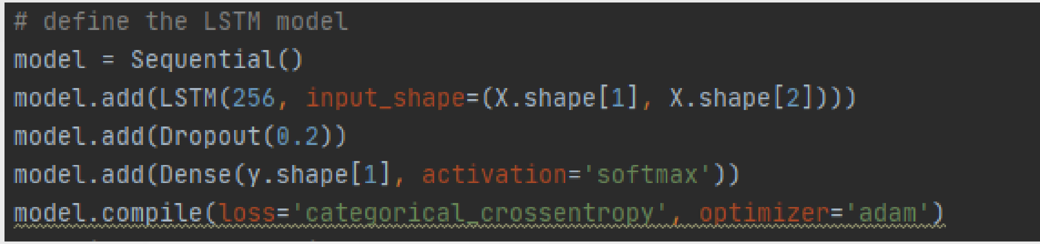
Training Model architecture

- Now every training pattern of the network is comprised of 100 time steps of one character (X) followed by one character output (y). All the characters have the chance to be learned from the preceded 100 character along the whole dataset except the first 100 characters.
- The total number of pattern will be the total number of the character -100 (the first 100 characters).
- After we get the sequences we have to make it suitable to be used by Keras by transferring the input sequences into the form of *[samples, time steps, features]*. Then rescale the integers to the range from 0 to 1. finally converting the output patterns into one encoding way (that make the network predict the probability of each of the 43 different characters).



```
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
```

Training Model architecture

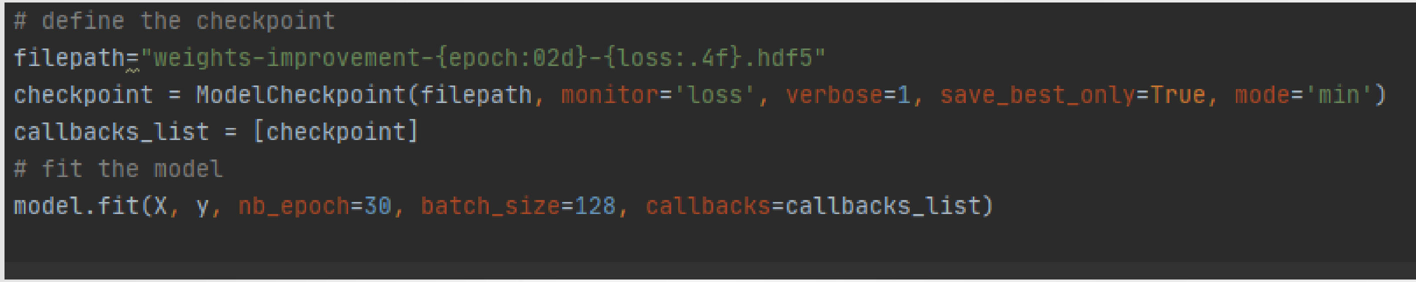


```
# define the LSTM model
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

➢ The next step is to define the LSTM model and we chose:

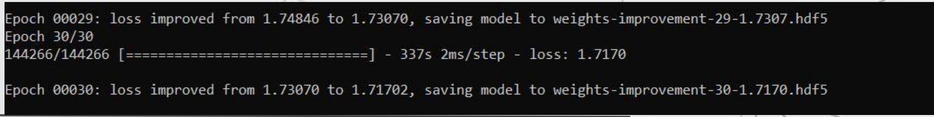
1. A singe hidden layer with 256 memory units
2. A dropout with rate of 0.2 which used to reduce the overfitting and improve the generalization error.
3. An output Dense layer with Softmax activation function (make the output probability prediction for each of the 43 characters between 0 and 1)
4. An optimization of the log loss (cross entropy) to classify the character from the 43 classes
5. ADAM optimization algorithm to speed the process.

Training Model architecture



```
# define the checkpoint
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]
# fit the model
model.fit(X, y, nb_epoch=30, batch_size=128, callbacks=callbacks_list)
```

- We used all the dataset for training to learn the probability of each character in a sequence.
- The aim of this model is to balance between the generalization and overfitting not to get the most accurate model.
- The next step is to use a model checkpointing to record all of the network weights to file each time an improvement in loss is observed at the end of the epoch.
- The last step is to set the number of epochs and the batch size.
- We will use the best checkpoint file (with the smallest loss value) to generate our finally text in the next chapter.



```
Epoch 00029: loss improved from 1.74846 to 1.73070, saving model to weights-improvement-29-1.7307.hdf5
Epoch 30/30
144266/144266 [==============================] - 337s 2ms/step - loss: 1.7170

Epoch 00030: loss improved from 1.73070 to 1.71702, saving model to weights-improvement-30-1.7170.hdf5
```

04

LSTM RNN Generating text Model

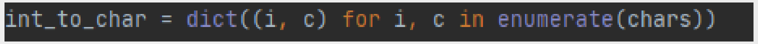
Generating Model architecture

- After we run the first python file and get the checkpoint files. The next step is to create another coding file to generate the new text sequences and that will be done by few steps:

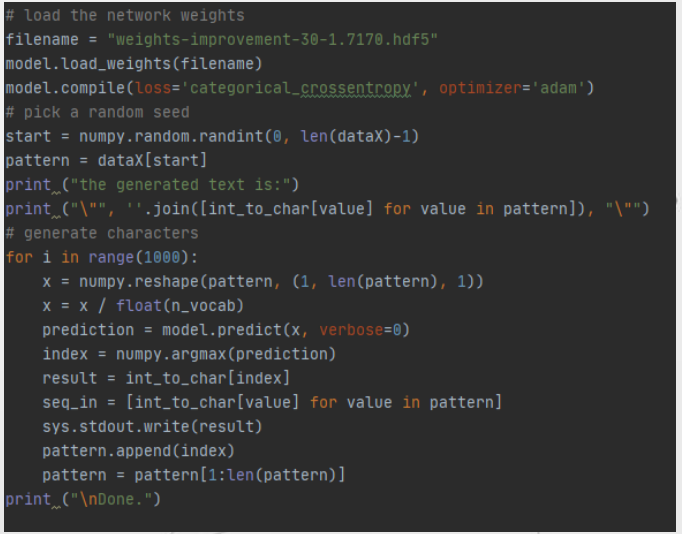
1- Using the same previous code from the beginning until defining the LSTM model with the single layer but with adding a new reverse mapping that convert the integers into characters.

2- Loading the network weights (the best checkpoint file with the smallest loss) that we get from the previous code.

3- Using the Keras LSTM model predictions by using randomly seed sequence as an input to the model to generate the next character then update the seed sequence by adding the generated character to the end of the seed after that removing the first character and continue until finishing the sequence (we used a sequence of 1000 character).



```
int_to_char = dict((i, c) for i, c in enumerate(chars))
```



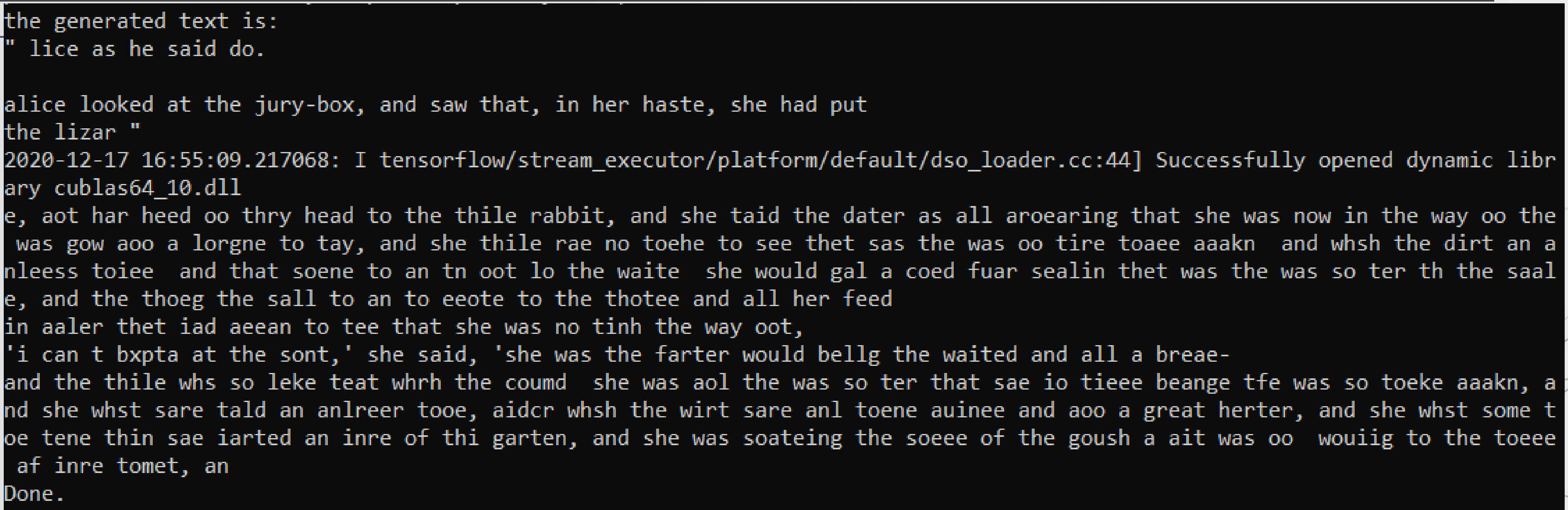
```
# load the network weights
filename = "weights-improvement-30-1.7170.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')
# pick a random seed
start = numpy.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print ("the generated text is:")
print ("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
# generate characters
for i in range(1000):
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(n_vocab)
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    sys.stdout.write(result)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]
print ("\nDone.")
```

05

Results and Discussion

Results

The following figure showing the results of 1-Alice's Adventures in Wonderland model with number of epoch=30, batch size=128
It shows the total number of the characters , vocab and patterns, then the generated text of this model.

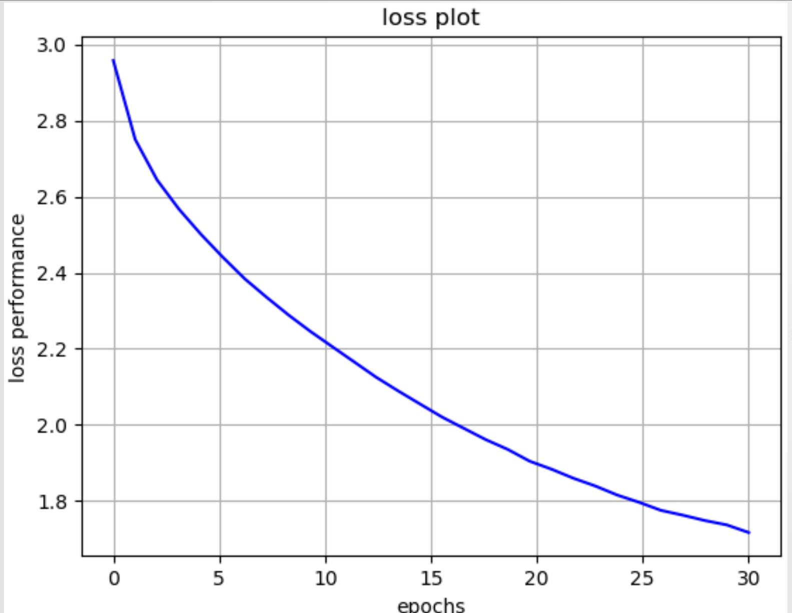


```
the generated text is:
" lice as he said do.

alice looked at the jury-box, and saw that, in her haste, she had put
the lizar "
2020-12-17 16:55:09.217068: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic libr
ary cublas64_10.dll
e, aot har heed oo thry head to the thile rabbit, and she taid the dater as all aroearing that she was now in the way oo the
 was gow aoo a lorgne to tay, and she thile rae no toehe to see thet sas the was oo tire toaee aaakn  and whsh the dirt an a
nleess toiee  and that soene to an tn oot lo the waite  she would gal a coed fuar sealin thet was the was so ter th the saal
e, and the thoeg the sall to an to eeote to the thotee and all her feed
in aaler thet iad aeean to tee that she was no tinh the way oot,
'i can t bxpta at the sont,' she said, 'she was the farter would bellg the waited and all a breae-
and the thile whs so leke teat whrh the coumd  she was aol the was so ter that sae io tieee beange tfe was so toeke aaakn, a
nd she whst sare tald an anlreer tooe, aidcr whsh the wirt sare anl toene auinee and aoo a great herter, and she whst some t
oe tene thin sae iarted an inre of thi garten, and she was soateing the soeee of the goush a ait was oo  wouiig to the toeee
 af inre tomet, an
Done.
```

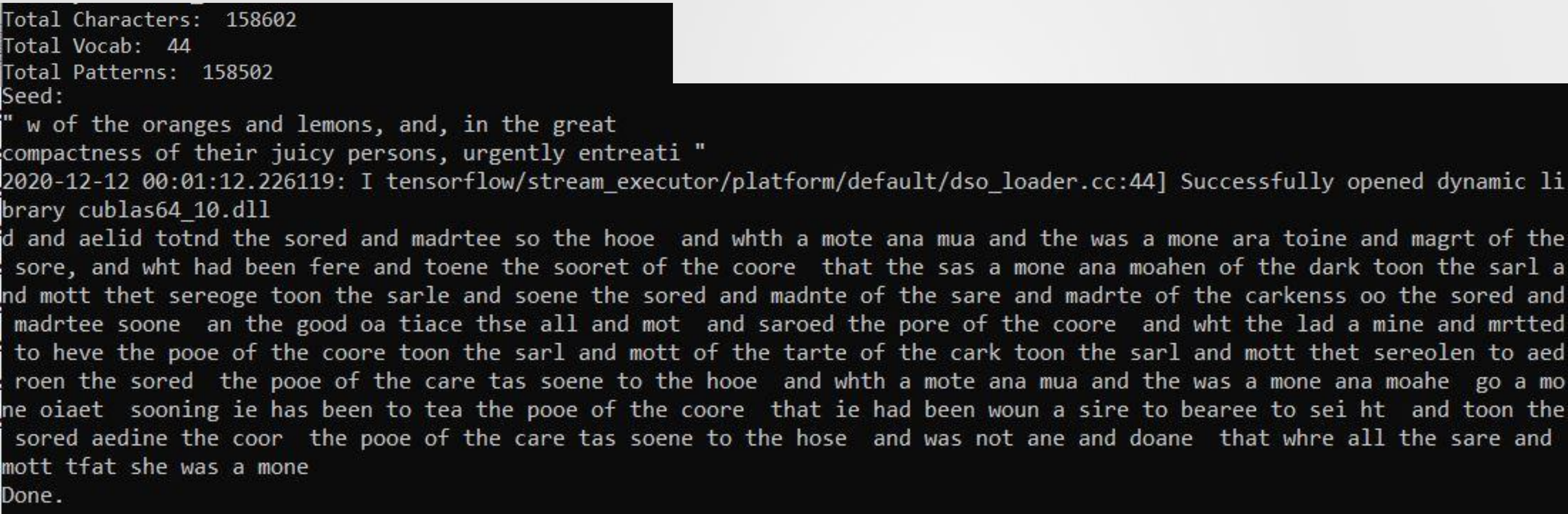
Results

The following figure showing the results of 1-Alice's Adventures in Wonderland model with number of epoch=30, batch size=128
It shows the total number of the characters , vocab and patterns, then the generated text of this model.



Results

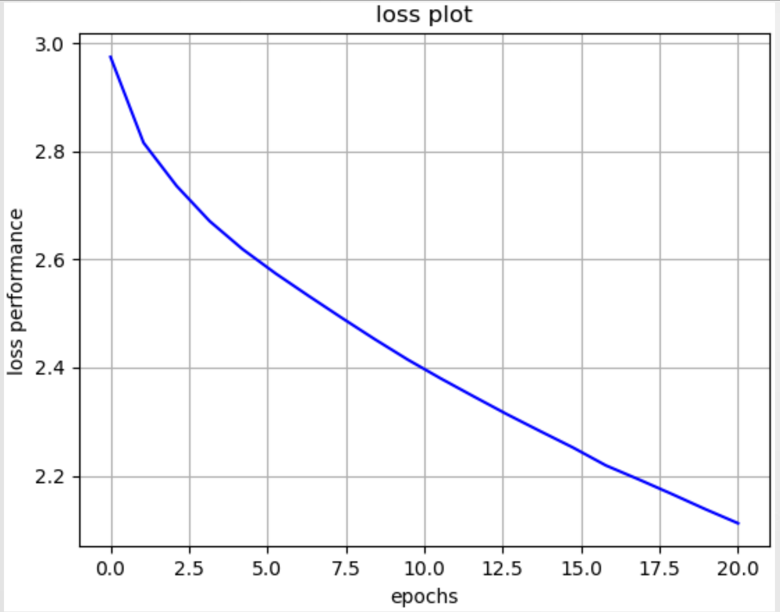
The following figure showing the results of 2- A Christmas Carol in Prose; Being a Ghost Story of Christmas.
With the same previous code and configurations but with 20 epochs.



```
Total Characters:  158602
Total Vocab:  44
Total Patterns:  158502
Seed:
" w of the oranges and lemons, and, in the great
compactness of their juicy persons, urgently entreati "
2020-12-12 00:01:12.226119: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic li
brary cublas64_10.dll
d and aelid totnd the sored and madrtee so the hooe  and whth a mote ana mua and the was a mone ara toine and magrt of the
 sore, and wht had been fere and toene the sooret of the coore  that the sas a mone ana moahen of the dark toon the sarl a
nd mott thet sereoge toon the sarle and soene the sored and madnte of the sare and madrte of the carkenss oo the sored and
 madrtee soone  an the good oa tiace thse all and mot  and saroed the pore of the coore  and wht the lad a mine and mrtted
 to heve the pooe of the coore toon the sarl and mott of the tarte of the cark toon the sarl and mott thet sereolen to aed
 roen the sored  the pooe of the care tas soene to the hooe  and whth a mote ana mua and the was a mone ana moahe  go a mo
ne oiaet  sooning ie has been to tea the pooe of the coore  that ie had been woun a sire to bearee to sei ht  and toon the
 sored aedine the coor  the pooe of the care tas soene to the hose  and was not ane and doane  that whre all the sare and
mott tfat she was a mone
Done.
```

Results

The following figure showing the results of 2- A Christmas Carol in Prose; Being a Ghost Story of Christmas.
With the same previous code and configurations.



Discussion

To improve the work we should add more layers, raise the number of epochs and decrease the batch size to give the network more of an opportunity to be updated and learn.

- As we can note about the generated texts.
- The characters are separated into word-like groups and most groups are actual English words (e.g. “the”, “of” and “was”), but many do not (e.g. “toeeeren”, “hooe” and “taede”).
- Some of the words in sequence make sense(e.g. “said alice to herself“), but many do not (e.g. “has been to tea“).
- The fact that this character based model of the book produces output like this is very impressive. It gives a sense of the learning capabilities of LSTM networks.

References

- https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/
- https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
- Sutskever, Ilya, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." ICML. 2011.

Thank you for your attention