Thomas Courtney

Artificial Intelligence

Professor Blossom

3 October 2021

Convolutional Neural Networks Assignment

Summary

The assignment called to design and implement a recurrent neural network that will produce headlines of four to eight words for articles. The user implements the RNN and is able to produce new headlines for the user.

Introduction

A recurrent neural network is one of the many different types of artificial neural networks. RNN’s create connections between nodes based off of a directed graph along a temporal sequence. RNN’s allow it to exhibit temporal dynamic behavior.

RNN’s are commonly seen in text sequencing, as shown in the example below. When writing different texts, these models predict what you are going to say next, based off the past examples you have typed.This can also be used in text summarization or chatbots.

In our example below, we will produce headlines for different articles based on a dataset that was previously published. The original dataset contained over 1,048,565 different headlines from articles contained in abc new articles. All headlines were taken from 2003 to 2016.

Body

"""

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Import Libraries Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

import numpy as np

from matplotlib import pyplot

import matplotlib.pyplot as plt

import tensorflow as tf

import tensorflow.keras

from numpy import array

from pickle import dump

from keras.preprocessing.text import Tokenizer

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Embedding

from tensorflow.keras.layers import Lambda

from tensorflow.keras.preprocessing.text import text\_to\_word\_sequence

from random import randint

from pickle import load

from tensorflow.keras.models import load\_model

from keras.callbacks import ModelCheckpoint

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import datetime

from nltk.tokenize import word\_tokenize

import pandas as pd

from nltk.tokenize import sent\_tokenize,word\_tokenize

from random import randint

from pickle import load

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing.sequence import pad\_sequences

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Parameters Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

in\_filename = 'abcnews-date-text.csv' #grab your file

num\_input\_words = 250 #set the number of input words

epochs = 100 #set the number of epochs

batch\_size = 128 #set the batch size

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Load Data Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

def load\_doc(in\_filename): # load in doc

file = open(in\_filename, 'r') # open the file

text = file.read() # read text

file.close() # close file

return text

#raw\_text = open(filename, encoding="utf8").read()

#raw\_text = raw\_text.lower()

doc = load\_doc(in\_filename) #load in the file

#lines = doc.split('\n')

#lines = doc.splitlines

lines = text\_to\_word\_sequence(doc) #set your variable

#print(lines)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Pretreat Data Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# encode the word sequence

tokenizer = Tokenizer()

#update vocabulary

tokenizer.fit\_on\_texts(lines)

#transform each text into a sequence of integers

sequences = tokenizer.texts\_to\_sequences(lines)

#definethe size of the vocabulary

vocab\_size = len(tokenizer.word\_index) + 1

# separate input and output

sequences = array(sequences)

#create the variables

X, y = sequences[:], sequences[:]

#make the y variable categorical

y = to\_categorical(y, num\_classes=vocab\_size)

#define the length of sequence

seq\_length = X.shape[1]

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Define Model Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#start the timer to see how long our model takes to run

start\_time = datetime.datetime.now() #start timing the model

model = Sequential() #define model as sequential

model.add(Embedding(vocab\_size,50,input\_length = seq\_length))#create a hidden embedding layer

model.add(LSTM(num\_input\_words, return\_sequences=True)) # LSTM layer 1

model.add(LSTM(num\_input\_words)) #LSTM layer 2

model.add(Dense(64, activation='tanh')) #hidden dense function with tanh

model.add(Dense(num\_input\_words, activation='relu')) #hidden dense function with relu

model.add(Dense(64, activation='tanh')) #hidden dense function with tanh

model.add(Dense(num\_input\_words, activation='relu')) #hidden dense function with relu

#model.add(Lambda(lambda x: x / 100))

model.add(Dense(vocab\_size, activation='softmax')) #initialize output Dense function

print(model.summary()) #print the model summary

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']) #compile the model

#save the weights from each of the epochs

filename = "weights.best.hdf5"

#set the break points to see if accuracy increased

checkpoint = ModelCheckpoint(filename, monitor='val\_accuracy', verbose=1, save\_best\_only=True, mode='max') #define the callback for fit function

callbacks\_list = [checkpoint]

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Train Model Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# fit model

history = model.fit(X, y,

validation\_split = .4,

batch\_size=batch\_size,

epochs=epochs, verbose = 1,

callbacks = callbacks\_list)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Show output Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#stop the timer and display how long it took to run the model

stop\_time = datetime.datetime.now()

print ("Time required for training:",stop\_time - start\_time) # save the model to file

model.save('model\_final.h5')

# save the tokenizer

dump(tokenizer, open('tokenizer.pkl', 'wb'))

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Show output Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

scores = model.evaluate(X, y, verbose=1)# evaluate model

print('Test loss:', scores[0]) # print the models accuracy and loss

print('Test accuracy:', scores[1])

#define the variables for each metric and store their values through each epoch

acc = history.history['accuracy']

#val\_acc = history.history['val\_acc']

loss = history.history['loss']

#val\_loss = history.history['val\_loss']

#show how many epochs there were

epochs = range(len(acc))

#plot the accuracy of each train and test dataset

plt.plot(epochs, acc, 'green', label='Training acc')

#plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training accuracy')

plt.legend()

plt.figure()

#plot the loss of each train and test dataset

plt.plot(epochs, loss, 'green', label='Training loss')

#plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training Loss')

plt.legend()

plt.show()

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Parameters Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

filename = 'abcnews-date-text.csv'

num\_words = random.randint(4,12)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Load Data Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# load doc into memory

def doc\_load(filename):

# open the file

file = open(filename, 'r')

# read the text

text = file.read()

# close file

file.close()

return text

# generate language model

def generate\_seq(model, tokenizer, seq\_length, seed\_text, n\_words):

result = list()

in\_text = seed\_text

# generate the number of words

for \_ in range(n\_words):

# encode

encoded = tokenizer.texts\_to\_sequences([in\_text])[0]

# sequences should be truncated

encoded = pad\_sequences([encoded], maxlen=seq\_length, truncating='pre')

# predict probabilities for each word

yhat = model.predict(encoded, verbose=0)

# y\_hat\_classes=np.argmax(yhat,axis=1)

# map predicted word index to word

out\_word = ''

for word, index in tokenizer.word\_index.items():

if index ==np.any(yhat):

out\_word = word

break

# append to input

in\_text += ' ' + out\_word

result.append(out\_word)

return ' '.join(result)

# load cleaned text sequences

doc = doc\_load(filename)

lines = doc.split('\n')

seq\_length = len(lines[0].split()) - 1

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Define Model Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# load the model

model = load\_model('model.h5')

# load the tokenizer

tokenizer = load(open('tokenizer.pkl', 'rb'))

# select a seed text

seed\_text = lines[randint(0,len(lines))]

print(seed\_text + '\n')

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

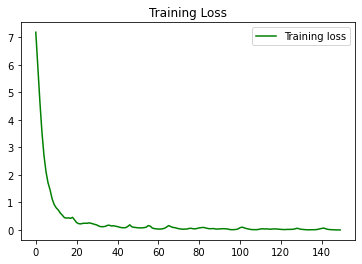
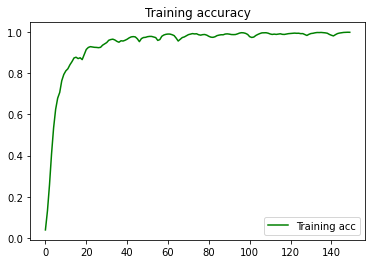
Show output Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

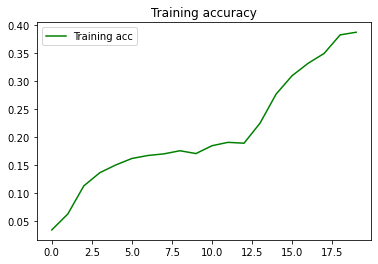
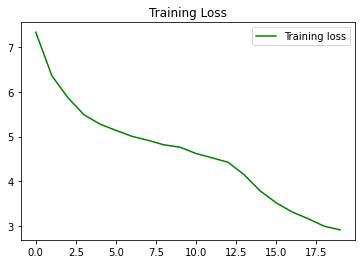
# generate new text

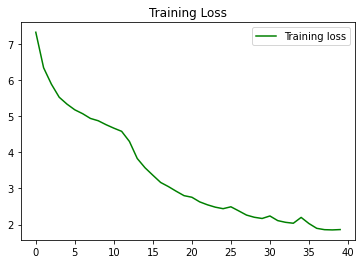
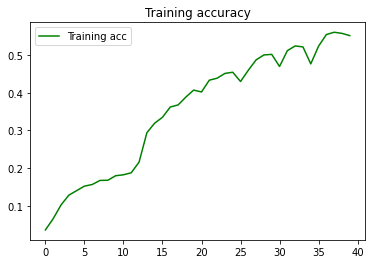
generated = generate\_seq(model, tokenizer, seq\_length, seed\_text, num\_words)

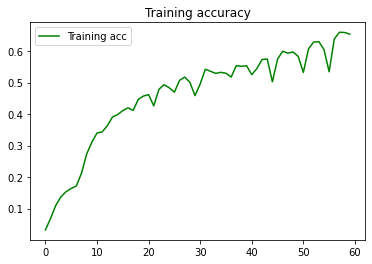
print(generated)

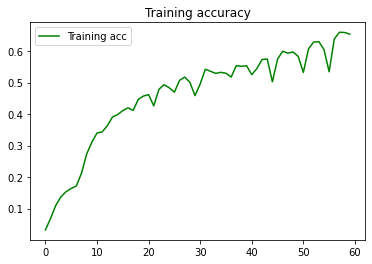
In the code above we implement RNN. We generate each text by word, train the RNN using the dataset. The model required few parameters to create a successful model. The first parameter is “in\_filename.” This file allows the model to be trained on common sequences of words. The next parameter is the number of input words. This is the number that the model is being trained on. The team determined 150 based on the outcome of the accuracy. In our model, we use two activation layers and two hidden layers. The hidden layer has two activation functions contained in the dense layer, relu and softmax. These were chosen over sigmoid, tanh and other examples because they improved accuracy the most. Below is the training accuracy and training loss for learning the data. As you will see, we kept getting the same words when we were generating text, so we went through multiple iterations of the model to try and avoid this. This includes changing the parameters, different dense layers and other items throughout the creation of the model.

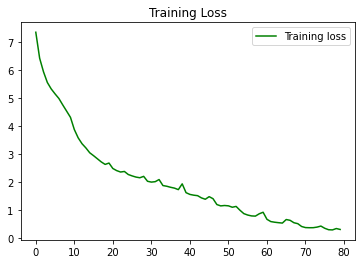
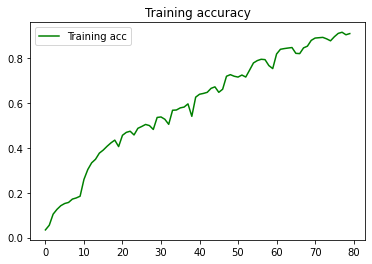
One thing that was really difficult about this assignment was that the output never seemed to improve based on any number of epochs. I continued to get the word “to” repeated over and over again. While training the model, there was no difference in the text generated, as shown in the table below. However, as expected the accuracy increases and the loss decreases depending on the number of epochs. At 150 Epochs, we get to 100% accuracy, as shown above. At around 50 Epochs, we stay at around 100% accuracy.







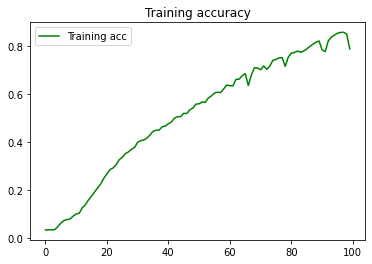




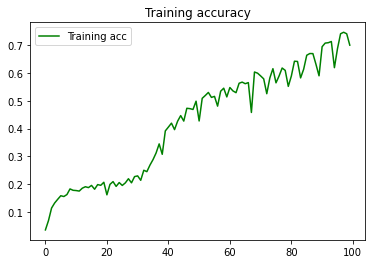
As mentioned previously, we continued to get results of “to” repeated over and over again. The team is unsure why this kept occurring as they spent over 30 hours trying to tune the model correctly. Below are the results.

| **Seed** | **Headlines** |
| --- | --- |
| slack thinking about getting reds on track not the | to to to to to to to to to |
| kato loses fight for life | to to to to to to to to |
| bendigo bombers boost membership | to to to to to |
| mp says nominating to chair hill probe | to to to to |
| doctors group questions planned medicare rebate | to to to to to to to to to to to to |
| rail move to disrupt services | to to to to to |
| tufnell ends controversial career | to to to to |
| call for permanent ban on duck quail shooting | to to to to to to to to |
| mayors urge abattoir drought aid | to to to to to to to to to |
| police to enforce easter trading laws | to to to to |
| mine leaders welcome push for exploration boost | to to to to |
| adelaide notch second nsl finals win | to to to to to to to |
| teen shows up pros to make final threesome in | to to to to to to to to to |
| well be back say beaten ferrari | to to to to to to to to to to to to |
| underground facilities found at baghdad airport us | to to to to |

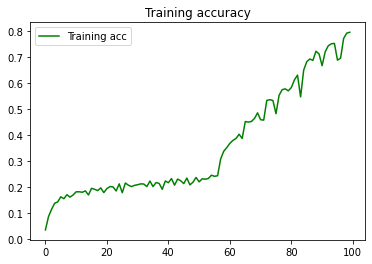
In testing, we dropped down the number of hidden units to 32 in testing. Dropping down the number of units decreased the testing accuracy to .78 and increased the loss to .987. It did not change our text generation however.



We changed the number of hidden dense layers in our model. Originally this model only had two hidden dense layers. We doubled this. This decreased our training accuracy down to .64 and our loss increased to 2.3. This did not change our text generation however.



We changed the sequence length of our models. First we doubled our sequence length, and then we cut it in half. layers in our model. Doubling this amount we saw a test loss to be 2.56 and the test accuracy is .79. This did not change our text generation however.



Through testing for a very long time, our outputs never really changed. The best headline was “to to to to to” and the funniest headline was “to to to to to to to to to to to to”

Conclusion

Overall, recurrent neural networks make the text generation process happen. This however takes a great amount of computational power to run the model. To do this, the user had to decrease the amount of data contained in the model. The user kept the original 7000 records only. In the real world where businesses are using computational neural networks, there is no fear that we will not have the correct amount of computational power.

Recommendations

Be cautious when using recurrent neural networks. No artificial intelligence platform will ever be completely accurate, so please consider before making decisions. Moving forward, I would like to be able to test this on a different computer so I could figure out why the text generation kept on repeating the same value

Appendix

The dataset was taken from Tensorflow and loaded into the code above.