## from tensorflow.keras.preprocessing.text import Tokenizer

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

## 

Link to notebook: <https://colab.research.google.com/drive/1ZYbE5rIzD8mCH3aWvGh1AFCjcUlYEBTw?usp=sharing>

## Get the data from .txt file

data = open('/content/irish-lyrics.txt').read()

#Read the contents of the file. Convert all to lowercase and return a list where each element is obtained by separation of the newline

#If the content of file is:

###This is a good day

###I want to go swimming today

###But it's still cold

#then you will get a list:

#['this is a good day' , 'i want to go swimming today' ,'but it's still cold'] after the lower and split

corpus = data.lower().split('\n')

## Corpus will be a list containing the sentences in the datasetTokenizer

Notebook here:

<https://colab.research.google.com/drive/1RSM82t50td6mADw3XY-jls5MKO9Cm2u8?usp=sharing>

sentences = ['I am a dog',

'This is a cat',

'I don"t keep pets']

tokenizer = Tokenizer(num\_words=100,oov\_token='<OOV>')

tokenizer.fit\_on\_texts(sentences)

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

1. Define the Tokenizer
2. Fit\_on\_text to get the encoding dictionary
3. Texts\_to\_seq to transform sentences to their encoded form

## Padding Sequence

padded = pad\_sequences(seq, maxlen = 10, padding = 'post' , truncating = 'post')

#padding = 'post' means you want the padding to be on the right (instead of the default 'pre' which is on the left)

#truncating = 'post' means that when words exceed max len, you remove the extra words on the Right (where you remove the words on the left for 'pre')

1. Pad\_sequences to get all sentences to be of same length

word\_idx = tokenizer.word\_index

print(word\_idx)

{'the': 1, 'and': 2, 'i': 3, 'to': 4}

* Word\_idx will be the encoding dict mapping the word to its encoded value

## Creating the n-grams

input\_seq = []

#Loop through all the lines in the corpus

for line in corpus:

#Tokenize the line in the corpus. Pass the 'line' as a list and zero index it so you can get just a list of all the words

#(This just formats the list to the way you want it to be)

tokenized = tokenizer.texts\_to\_sequences([line])[0]

#You want to create an n-gram, so if the sentence is 'I am very happy'

#The n-gram will be:

#[I am]

#[I am very]

#[I am very happy]

#With the words replaced by their tokenized form

#Hence, I loop through the length of the tokenized line, starting from 1

#(because later I need to remove the last word to be my label,hence my n-gram needs to have a minimum of 2 words;so there is at least one word left when one word is taken out for the label)

for i in range(1,len(tokenized)):

#Create the n\_gram that takes the words all the way to (i+1)

#If the line has 4 words, it will create 3 n-grams

n\_gram = tokenized[:i+1]

#add this n\_gram to the input\_seq list

input\_seq.append(n\_gram)

* When doing texts\_to\_sequence, pass in the line as a list [‘this is a line’] instead of just a raw string ‘this is a line’
* Also remember to index the first element texts\_to\_sequences([line])[0] if not you will get a 2d array [[1,45,4,67]] instead of a 1d array [1,45,4,67]
* Using the tokenized sentence, create the n-grams by looping through the encoded sentence.
* If the length is 5, then you will get i = 1,2,3,4
* So, n\_gram will be
* Tokenized[:2] which will be first 2 words
* tokenized[:3 ] which will be first 3 words
* tokenized[:4] which will be first 4 words
* tokenized[:5] which will be all 5 words

## Pad the n-grams

#Pad the sequence

#Loop through all the lists in input\_seq and find the max length

max\_len = max([len(x) for x in input\_seq])

#Pad the lists with the max\_length of the lists. Use 'pre' padding (which is default but indicated here to make things clearer)

padded = pad\_sequences(input\_seq,maxlen=max\_len,padding='pre')

print(padded[:20])

* Apply padding to n-grams

## 

## Get x and y data (Split + OHE)

padded = np.array(padded)

#Numpy indexing. Index is in the form [first\_dim, second\_dim)

#[ <FIRST DIM> [<SECOND\_DIM>], [<SECOND\_DIM>], [<SECOND\_DIM>] <FIRST\_DIM> ]

# Each individual list [0,34,56,7] will be indexed by the first dimension [:,]

# Each word in the individual list will be indexed by the second dimension [,:]

#Since you want to remove the last element for all the lines in padded, and keep that last element as the label

x\_train = padded[:,:-1]

#I call this label first, not y\_train as it still has to undergo OHE

label = padded[:,-1]

#One Hot Encode the labels. There will be vocab\_size number of classes as you are predicting what the next word will be,choosing from all the words in the corpus (which is vocab\_size unique words)

#Hence, you OHE with num\_classes = vocab\_size (number of words you tokenized) + 1 so that you can predict what word will come after the next, during text generation

y\_train = tf.keras.utils.to\_categorical(label,num\_classes=vocab\_size+1)

* X\_train will be the n\_gram containing all but the last word
* The corresponding y\_train data will be the last word for that n\_gram
* When OHE, the num\_classes will be the total number of words in your vocabulary (Since the predicted word will be from this dictionary)
* Y\_train will be the One Hot Encoded version of the corresponding encoded integer. (So if y\_train = 3, it will be [0,0,1,0,0,....]

## Define Model

#This is the vocab\_size to be used for training. It's +1 because you follow the vocab\_size you passed in to the one hot encoder (To account for the OHE producing a 0 class)

total\_words = vocab\_size + 1

model = Sequential()

#input\_length is -1, because you removed the last word to be the label. The input\_length here still included the label together with the words

model.add(Embedding(input\_dim=total\_words,output\_dim=100,input\_length=max\_len-1))

model.add(Bidirectional(LSTM(150)))

#This is the prediction layer. You are choosing the predicted word (Which is a word chosen from the total\_words)

#The prediction will be in the form of a OHE vector, which has the class of the word it predicts to be next

model.add(Dense(total\_words,activation='softmax'))

adam = Adam(lr=0.01)

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

history = model.fit(x\_train,y\_train,epochs=100, verbose=1)

* Use an Embedding -- Bidirectional LSTM -- Dense

## Plot graph of accuracy and loss

import matplotlib.pyplot as plt

def plot(history,string):

fig = plt.figure()

y = history.history[string]

plt.plot(y)

plt.xlabel('Epochs')

plt.ylabel(string)

fig.suptitle(string,fontsize=20)

plt.show()

plot(history,'accuracy')

plot(history,'loss')

* This function will take the history (trained model) and plot a graph of the corresponding metric (Accuracy or loss)

## Make prediction

own\_text = input('What is your starting line:')

next\_words = int(input('How many words to add on?:'))

for \_ in range(next\_words):

#Get the tokenized version of your text

#Since you didn't include OOV, hence the unknown words will just be excluded (Dropped out completely)

own\_tokenized = tokenizer.texts\_to\_sequences([own\_text])[0]

#Must pass in a list to pad\_sequences. Pad it using the same way you padded the training data

own\_padded = pad\_sequences([own\_tokenized],maxlen=max\_len-1,padding='pre')

#Predict the class of the next word, given the current sentence (own\_padded)

predicted\_class = model.predict\_classes(own\_padded)

#Initialize the predicted word to an empty string first

predicted\_word = ''

#Get the key and value pair of the dictionary using .items()

for word, token in word\_idx.items():

#If this token is the same as the predicted class

if token == predicted\_class:

#Set the predicted word as the word which has this token

predicted\_word = word

#Stop after the predicted\_word has been found once

break

#Loop through this for 'next\_word' number of times

#(if it's 10, you loop through this 10 times to get 10 extra words predicted)

own\_text += ' ' + predicted\_word

#Prints the text where all predictions have been made and all predicted words appended to the end of the sentence

print(own\_text)