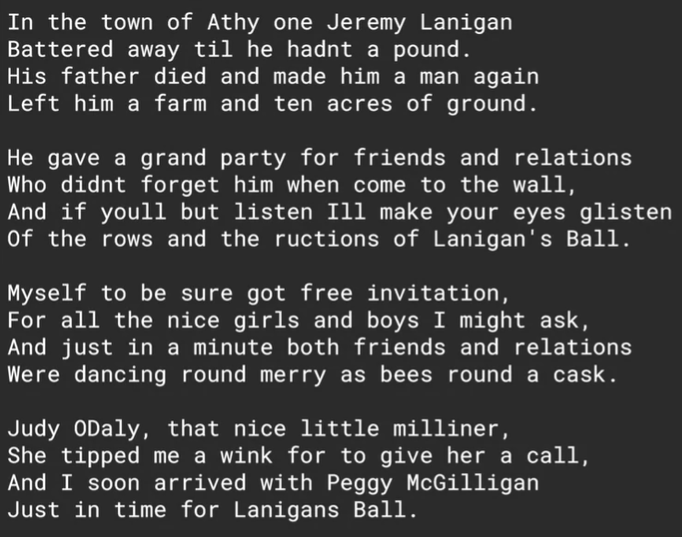
* The week is booked for Text generation and how we can build tensorflow models to generate texts

## Text generation principle

* Text generation isn’t finally a very complex problem like it appeared at the first time if we look at it as a prediction task , where given as in input X : a sentence ( a sequence of words ) : try to predict the next word Y . It’s a multi classification problem where we are gonna pick the word with the biggest softmax value.
  + So as example , if we already know that “Twinkle , Twinkle little Star” is a valid sentence , wee can consider “Twinkle , Twinkle little” as X and its next word “Star’ as the expected output . In the same context we can consider “Twinkle , twinkle” as X and “Little” is its Y .

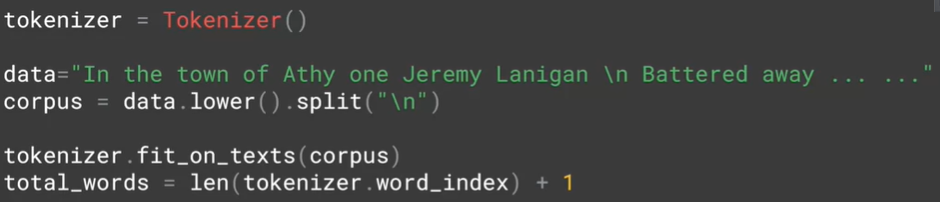
## Preparing the training Dataset

### Choosing the corpus of text to train on:



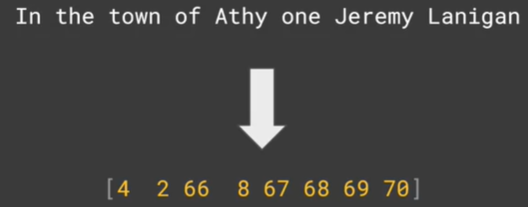
In the course example , it was an Irish song

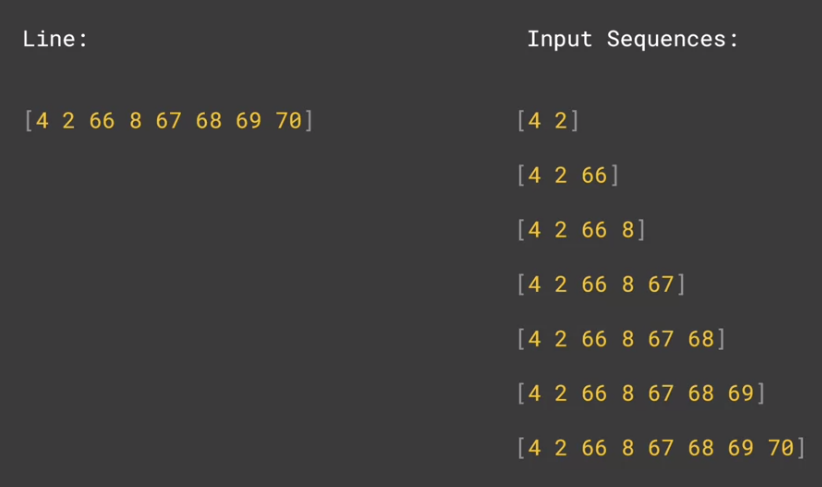
### Importing it and splitting to sentences to fit the tokenizer :

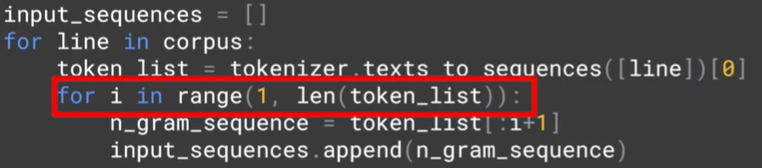


* For the third line , we called lower method to make our training non—case sensitive and the we called split(‘/n’) to have array each element is a sentence from this song
* In the last line , we added 1 to count the <OOV> token

### Generating from each sentence many training data :

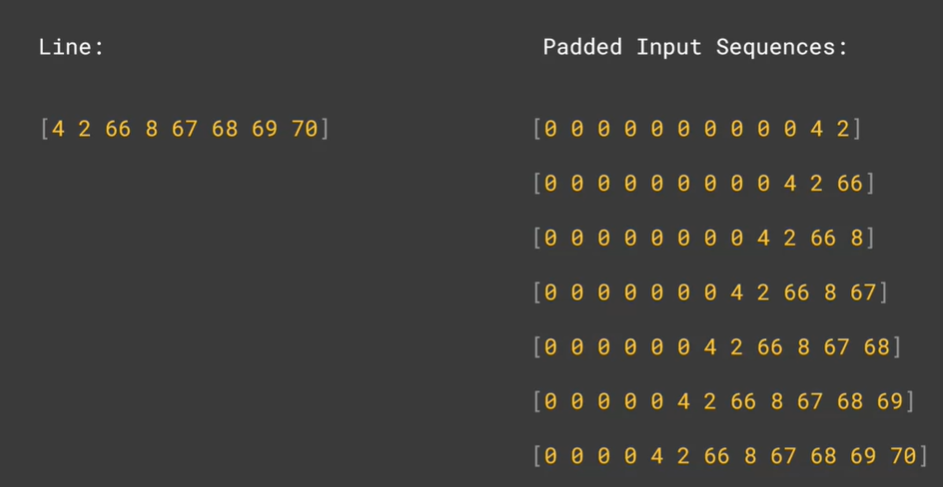


* Let’s say for example this is the array of index words corresponding to the first sentence
* Our goal is to make “the” is the generated word from “In” and “town” is the generated word from “In the” , ….. , and “Lanigan” is the generated word from the input “In the town of Athy one Jeremy”
* First of all , we are going to create number of n-grams sequences, namely the first two words in the sentence or one sequence, then the first three are another sequence etc , and to do so : the code below makes this process possible :

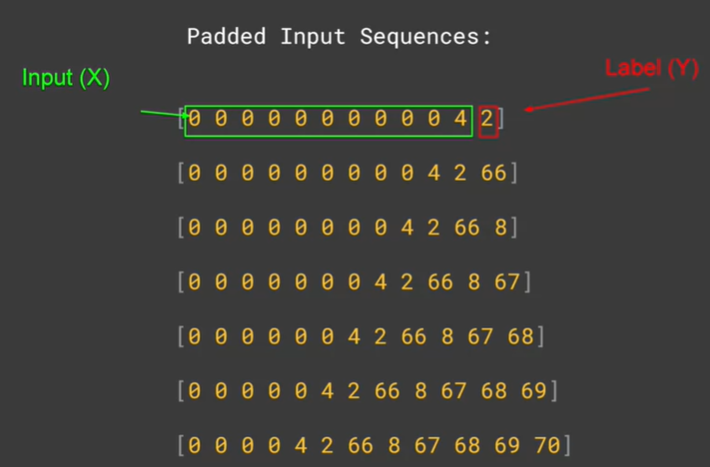


* For each sentence :
  + We are going to append to the training data : a sentence containing only the first two words , then the sentence containing only the first three words , …… , then finally the sentence containing the whole words

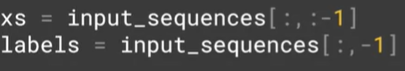
#### Padding of course is the next step , but it must be PRE



* Of course , we need to do padding process to unify the input shape of our model , the max\_length must be equal to the longest sentence length .
* the padding must be **pre** , it’s important so it’s easy to extract the output Y from each sentence which is the last item In each vector :



#### Extracting the Labels and the inputs from the padded sequences:



* The code showed how can we slice the last column to get the labels

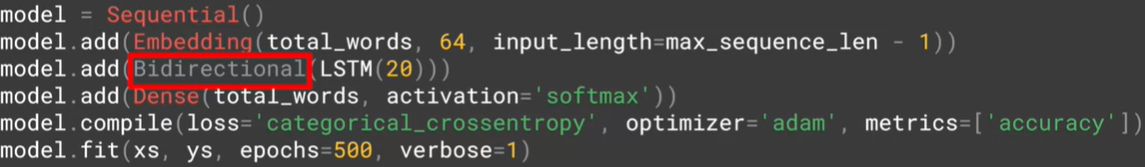
#### It’s a classification problem , we should call to categorical on the labels !



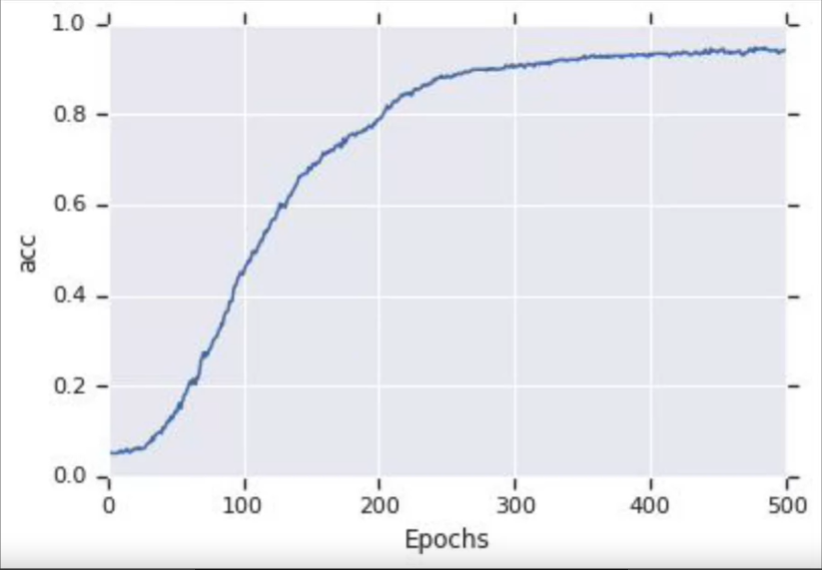
* The role of to\_categorical function is to transform the index of the label ( of the generated word ) into one-hot vector its diamneison equals to the vocabulary size
* This is important , because in the last layer of our model we are gonna have a dense layer , each word must have its own unit which is gonna triggered for the most suitable word to generate : so the number of units in the last layer is equal to vocabulary size , with softmax function as activation function of course
* The image below makes everything clear :



#### Finally : the model

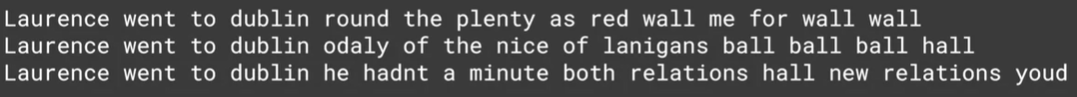


* Nothing surprising in the final model now
  + The embedding layer will have as a dimension vocab\_size\*64\*padded\_sequence\_length
    - 64 is the embedding vector dimension , it’s the only hyper parameter we can tweak here
  + The Bidirectional LSTM layer is the next layer
    - It’s logic to use LSTM in these kind of problems to memorize the whole sentence context while generating the next word
    - It’s important to be bidirectional to get the context from the previous and the next of the words to generate while training the model
  + The output layer is a dense layer with softmax as activation function since it’s a multi classification problem
    - The number of units must be equal to the vocabulary sizes because it’s the total number of our classes which defines which word to generate
  + The loss is of course ‘categorical\_crossentropy’ dur to the nature of the problem : multi classification
  + The Adam optimizer seems the best optimizer of generating sequences tasks due to mathematical reasons
* An interesting thing to note is the huge number of epochs we specified : 500 and this is because this kind of models takes a lot of epochs to converge , the graph below show the progression of the training accuracy during the 500 epochs :



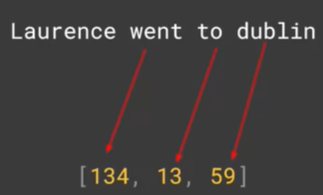
### Some examples of the generating sequences :

* We specified here “Laurence went to Dublin” as the start of our sequences ( we called it seed\_text ) and we told the model to generate the next 10 words



### How is that possible ? generating word by word

#### Tokenize the seed text :

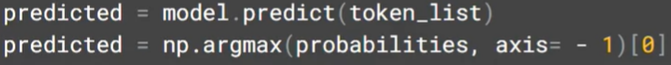


* Laurence hasn’t a corresponding index because it’s unknown word

#### The pre-padding :

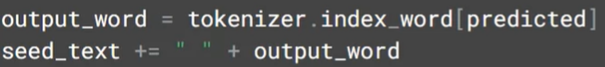


#### Predicting the next word:

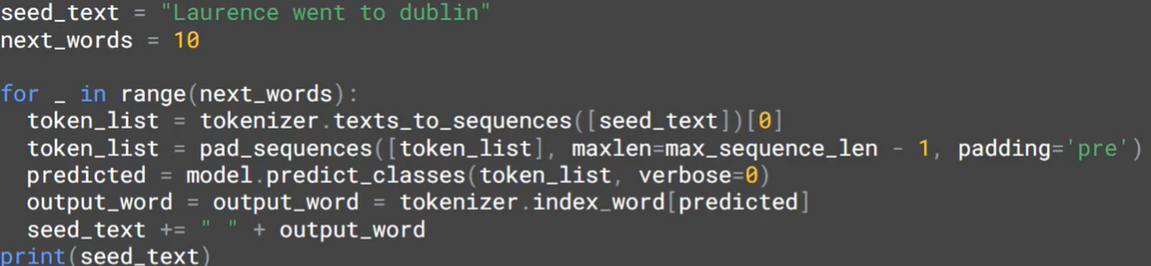


* We call the predict method in the seed\_tokenized\_pre-papadded sequence
* We took the word with the biggest softmax probability : it’s the next generated word !

#### Adding the generated word to the seed text :



#### Repeating the whole process in a loop with number of iterations equals to the desired final length :



## The generating sequence tasks aren’t really so accurate:



* Here is an example of a generated sequence by the model
* The reason of that is because the more we generated words after the base sentence : the more we would have less certainty about the correctness of the generated word