Next word prediction using LSTM RNN

1. Data collection :nltk.download(‘gutenberg’) -🡪shakespeare-hamlet
2. Intialize tokenizer(keras pre processing text)🡪 tokenizer.fit\_on([text]) 🡪 text is “hamlet.txt”
3. Total\_words = len(tokenizer.word\_index)🡪 tokenizer.word\_index( gives me index for each word in vocab)
4. Creating an input\_sequence using texts\_to\_sequence for each line of text( on \n). Then add a pad sequence, so that all sent are equal len.
5. X= leaving last element , Y = last element ,training and test 🡪 since this is next word prediction unlike sentiment analysis , we cant keep just the last word as one number, we have to match it to embeddings of other words, hence we add zeroes( as in whereever that index id present will be 1, rest will be 0)
6. y=tf.keras.utils.to\_categorical(y,num\_classes=total\_words)
7. total\_words=len(tokenizer.word\_index)+1
8. model=Sequential()
9. model.add(Embedding(total\_words,100,input\_length=max\_sequence\_len-1))
10. model.add(LSTM(150,return\_sequences=True))
11. model.add(Dropout(0.2))
12. model.add(LSTM(100))
13. model.add(Dense(total\_words,activation="softmax"))

100 = dimension of output vector input length( max length of one sentences) , max\_sequence\_len in here could be ki how much of sentence len of max could be passed as input for next word pred.

**1. Understanding y = tf.keras.utils.to\_categorical(y, num\_classes=total\_words)**

The line y = tf.keras.utils.to\_categorical(y, num\_classes=total\_words) converts the target variable (y) into a one-hot encoded format. Here’s why and how it’s done:

* **Purpose of y**: In language modeling, each input sequence corresponds to a target word (i.e., the word that follows the sequence in the text). y originally contains integer indices, where each index corresponds to a specific word in the vocabulary.
* **One-Hot Encoding Explanation**: One-hot encoding is a common technique used in classification problems, where the target variable is represented as a binary vector with a length equal to the total number of classes (total\_words in this case). Each element in the vector is 0 except for the index corresponding to the target class, which is set to 1.

So, if the y variable originally contains an integer (say, 4), this line converts it into a vector like [0, 0, 0, 0, 1, 0, ...], where only the 5th position (index 4) is 1, indicating that this position represents the target word for that sequence.

* **Why One-Hot Encoding for y?** In this context, one-hot encoding is used because this is a multi-class classification problem where the model needs to predict one of total\_words classes. Here, each word in the vocabulary is treated as a separate class. Using one-hot encoding helps the model understand that the target is a particular word out of the possible total\_words choices.

**2. Understanding predict\_next\_word Function**

The predict\_next\_word function is designed to take the trained model, the tokenizer, a text input, and the maximum sequence length as inputs. It then predicts the next likely word in the sequence. Let’s go through each line:

python

Copy code

def predict\_next\_word(model, tokenizer, text, max\_sequence\_len):

* **Function Definition**: This function takes four arguments:
  + model: the trained LSTM/GRU model used to predict the next word.
  + tokenizer: the tokenizer that was used to tokenize and convert words into integers.
  + text: the current input sequence (e.g., "To be or not to be") for which we want to predict the next word.
  + max\_sequence\_len: the length of the input sequences used during training, helping to ensure the input to the model has the correct shape.

python

Copy code

token\_list = tokenizer.texts\_to\_sequences([text])[0]

* **Tokenizing the Input Text**: This line converts the input text into a list of integers, where each integer represents a word in the vocabulary. The tokenizer used here is the same one used in training, so it should convert each word in text to its respective index in tokenizer.word\_index.

python

Copy code

if len(token\_list) >= max\_sequence\_len:

token\_list = token\_list[-(max\_sequence\_len-1):] # Ensure the sequence length matches max\_sequence\_len-1

* **Ensuring Correct Sequence Length**: If the tokenized token\_list is longer than max\_sequence\_len - 1 (the input length the model expects), we truncate it from the beginning to match this length. This ensures that the input sequence length is consistent with the training data, which helps the model generalize better.

python

Copy code

token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_len-1, padding='pre')

* **Padding the Sequence**: This line pads token\_list to ensure it matches the input shape required by the model (i.e., max\_sequence\_len - 1). Padding is done at the beginning (padding='pre') so that the sequence has the required length, even if it was shorter initially.

python

Copy code

predicted = model.predict(token\_list, verbose=0)

* **Making a Prediction**: Here, the model makes a prediction on token\_list. Since the model is trained to predict the probability distribution of the next word over all possible words, predicted will contain a probability score for each word in the vocabulary (each of total\_words classes).

python

Copy code

predicted\_word\_index = np.argmax(predicted, axis=1)

* **Finding the Word Index with Highest Probability**: Using np.argmax, this line extracts the index of the word with the highest probability from the predicted output. This is the index of the word that the model considers most likely to follow the input sequence.

python

Copy code

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

if index == predicted\_word\_index:

return word

* **Mapping the Index Back to the Word**: This loop goes through tokenizer.word\_index to find the actual word corresponding to predicted\_word\_index. When it finds a match, it returns the word as the predicted next word.

python

Copy code

return None

* **Fallback Return**: If for some reason the predicted index doesn’t match any word in the tokenizer (which is rare and unlikely here), the function returns None.