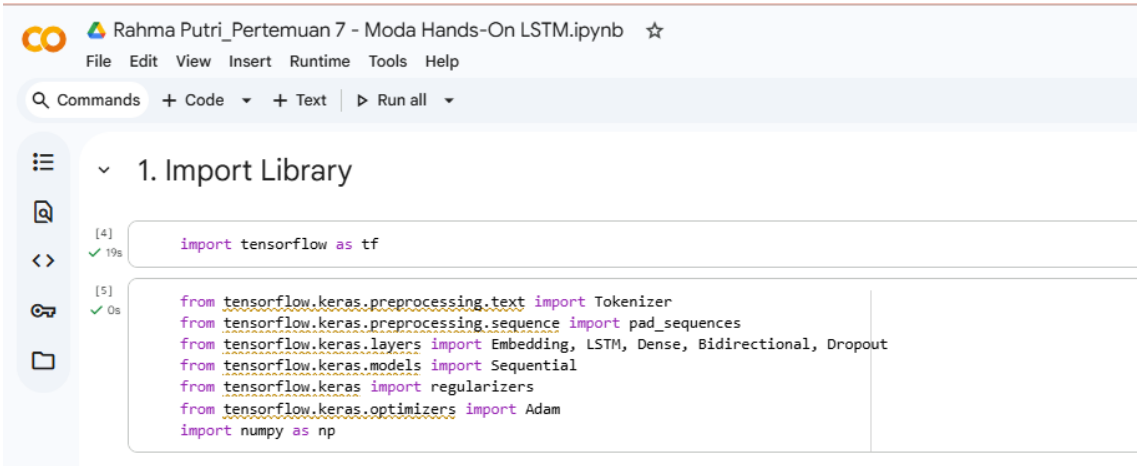
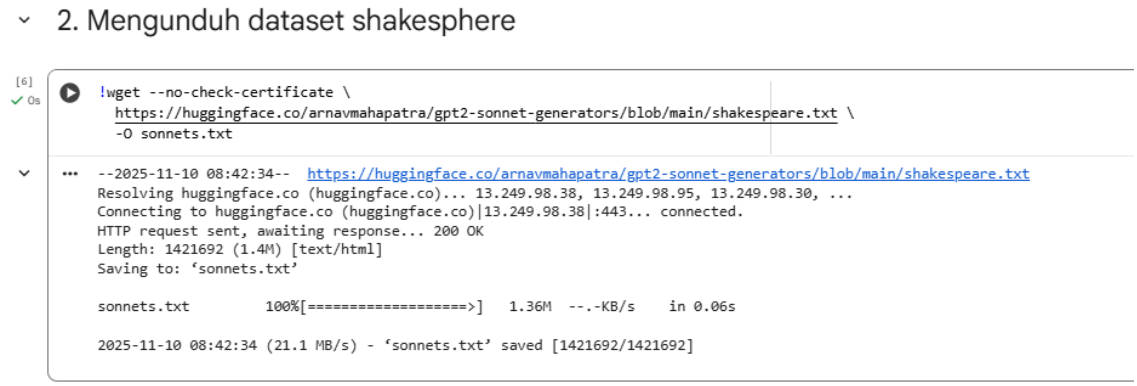
NAMA : RAHMA PUTRI PRABOWO

NPM : 11122170



1. Import Library

Pada tahap ini dilakukan proses *import* berbagai library yang dibutuhkan untuk membangun model *Recurrent Neural Network (RNN)* berbasis LSTM. Library **TensorFlow** dan **Keras** digunakan untuk membuat dan melatih model jaringan saraf. Modul **Tokenizer** dan **pad_sequences** berfungsi untuk memproses teks menjadi angka agar dapat dibaca oleh model. Sementara itu, **Embedding**, **LSTM**, **Dense**, **Dropout**, dan **Bidirectional** merupakan komponen jaringan yang akan digunakan dalam arsitektur model. Library **NumPy** digunakan untuk melakukan operasi numerik, sedangkan **Adam** digunakan sebagai algoritma optimisasi selama proses pelatihan.



2. Mengunduh Dataset Shakespeare

Langkah ini bertujuan untuk mengunduh kumpulan teks karya William Shakespeare yang akan digunakan sebagai data pelatihan. Dataset diunduh dari *Hugging Face Hub* menggunakan perintah wget, lalu disimpan dalam file bernama **sonnets.txt**. File ini berisi kumpulan puisi atau soneta yang akan digunakan untuk melatih model dalam memprediksi kata berikutnya berdasarkan urutan teks. Data tersebut menjadi sumber utama dalam proses pembelajaran model bahasa.

3. Mendefinisikan Tokenizer dan Menyiapkan Training Data

Langkah selanjutnya yaitu melakukan proses tokenisasi dan menyiapkan data yang akan dilatih. tokenisasi adalah proses untuk membagi teks yang dapat berupa kalimat, paragraf atau dokumen, menjadi token-token/bagian-bagian tertentu. Pada proses ini suatu kata yang ada pada data sheet akan disimbolkan dengan angka secara acak sampai jumlah kata yang ada pada datasheet tersebut. Pada datasheet ini jumlah katanya yaitu sampai 2900. Contohnya yaitu angka 1 didefinisikan untuk kata "and", 2 untuk kata "the" dan seterusnya. Berikut adalah kodingan serta output dari proses tokenizer.

]:
```
tokenizer = Tokenizer()

data = open('sonnets.txt').read()

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

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

print(tokenizer.word_index)
print(total_words)
```

```
{'and': 1, 'the': 2, 'to': 3, 'of': 4, 'my': 5, 'i': 6, 'in': 7, 'that': 8, 'thy': 9, 'thou': 10, 'with': 11, 'for': 12,
'a': 13, 'but': 14, 'love': 15, 'is': 16, 'thee': 17, 'not': 18, 'me': 19, 'so': 20, 'you': 21, 'all': 22, 'be': 23, 'hi
s': 24, 'when': 25, 'as': 26, 'which': 27, 'your': 28, 'this': 29, 'it': 30, 'doth': 31, 'by': 32, 'or': 33, 'do': 34, 'f
rom': 35, 'then': 36, 'on': 37, 'no': 38, 'are': 39, 'their': 40, 'shall': 41, 'what': 42, 'have': 43, 'mine': 44, 'tim
e': 45, 'if': 46, 'sweet': 47, 'more': 48, 'beauty': 49, 'art': 50, 'than': 51, 'where': 52, 'nor': 53, 'can': 54, 'shoul
d': 55, 'will': 56, 'now': 57, 'o': 58, 'how': 59, 'they': 60, 'he': 61, 'yet': 62, 'hath': 63, 'eyes': 64, 'make': 65,
'thine': 66, 'fair': 67, 'eye': 68, 'him': 69, 'one': 70, 'still': 71, 'like': 72, 'being': 73, 'every': 74, 'own': 75,
```

```
input_sequences = []

for line in corpus:
  token_list = tokenizer.texts_to_sequences ([line])[0]
  #print("LIST"+str(token_list))
  for i in range(1, len(token_list)):
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)


##pad sequences
max_sequence_len = max([len(seq) for seq in input_sequences])

print(max_sequence_len, total_words)

input_sequences = np.array(pad_sequences(input_sequences, padding='pre', maxlen=max_sequence_len))

#create predictors and labels
xs, labels = input_sequences[:,:-1], input_sequences[:,-1]

ys = tf.keras.utils.to_categorical (labels, num_classes=total_words)
```

3. Mendefinisikan Tokenizer dan Menyiapkan Data Training

Tahap ini adalah proses *tokenisasi*, yaitu mengubah teks mentah menjadi urutan angka. Setiap kata unik pada dataset diberi indeks numerik menggunakan objek **Tokenizer**, sehingga model dapat memahami teks dalam bentuk numerik. Setelah itu, dibuat *n-gram sequences* untuk melatih model agar bisa mempelajari hubungan antar kata. Proses *padding* digunakan untuk menyamakan panjang setiap urutan agar dapat diproses oleh jaringan. Data kemudian dipisahkan menjadi dua bagian, yaitu **xs (fitur/predictor)** dan **ys (label/target)**, di mana model akan belajar memprediksi kata berikutnya berdasarkan kata sebelumnya.

4. Mendefinisikan Arsitektur Model

Langkah selanjutnya yaitu mendefinisikan arsitektur , pada tahap ini kita menentukan berapa total kata yang ada pada artikel tersebut kemudian menentukan urutan setiap kata yang ada pada artikel tersebut. Berikut adalah kodingan serta output dari mendefinisikan arsitektur modelnya.

In [15]:
```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150, return_sequences=True)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(96)))
model.add(Dense(total_words//2, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(total_words, activation='softmax'))

print(model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is de
precated. Just remove it.
  warnings.warn(
Model: "sequential_6"
```

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	?	0 (unbuilt)
bidirectional_12 (Bidirectional)	?	0 (unbuilt)
dropout_6 (Dropout)	?	0 (unbuilt)
bidirectional_13 (Bidirectional)	?	0 (unbuilt)
dense_12 (Dense)	?	0 (unbuilt)
dense_13 (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

4. Mendefinisikan Arsitektur Model

Tahap ini mendesain struktur jaringan RNN menggunakan model **Sequential** dari Keras. Lapisan pertama adalah **Embedding layer** yang mengubah indeks kata menjadi vektor representasi numerik. Kemudian terdapat dua lapisan **Bidirectional LSTM** yang memungkinkan model membaca urutan kata dari dua arah sekaligus, sehingga konteks antar kata dapat dipahami lebih baik. Lapisan **Dropout** digunakan untuk mencegah *overfitting*, sementara lapisan **Dense** dengan fungsi aktivasi *relu* dan *softmax* digunakan untuk menghasilkan probabilitas kata berikutnya. Arsitektur ini memungkinkan model mempelajari pola bahasa dari teks Shakespeare dengan efisien.

5. Training Data

```
[16]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
      history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
Epoch 1/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 62s 143ms/step - accuracy: 0.0237 - loss: 7.7045
Epoch 2/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 55s 145ms/step - accuracy: 0.0209 - loss: 6.4439
Epoch 3/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 80s 139ms/step - accuracy: 0.0265 - loss: 6.3765
Epoch 4/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 82s 138ms/step - accuracy: 0.0268 - loss: 6.3082
Epoch 5/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 82s 137ms/step - accuracy: 0.0316 - loss: 6.2169
Epoch 6/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 83s 139ms/step - accuracy: 0.0419 - loss: 6.0480
Epoch 7/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 83s 142ms/step - accuracy: 0.0393 - loss: 5.9812
Epoch 8/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 83s 147ms/step - accuracy: 0.0433 - loss: 5.9233
Epoch 9/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 78s 137ms/step - accuracy: 0.0506 - loss: 5.7860
Epoch 10/10
380/380 ━━━━━━━━━━━━━━━━━━━━ 81s 135ms/step - accuracy: 0.0470 - loss: 5.7064
```

5. Training Data

Pada tahap ini model dikompilasi menggunakan fungsi *loss* **categorical_crossentropy**, optimisasi **Adam**, dan metrik **accuracy** untuk mengukur performa. Selanjutnya, model dilatih menggunakan data input dan target hasil tokenisasi. Setiap *epoch* mewakili satu kali proses pembelajaran seluruh dataset. Selama pelatihan, nilai *loss* menurun dan *accuracy* meningkat, yang menunjukkan bahwa model mulai memahami hubungan antar kata dalam teks. Hasil pelatihan inilah yang nantinya digunakan untuk memprediksi kata berikutnya.

6. Membuat Perintah Untuk 100 Kata Selanjutnya

Langkah selanjutnya yaitu membuat perintah untuk RNN supaya dapat memprediksi 100kata selanjutnya pada artikel yang ingin dibuat berdasarkan datasheet yang sudah diunduh pada langkah sebelumnya. Pada langkah ini RNN akan dipancing untuk memprediksi kataselanjutnya berdasarkan seed textyang sudah ditentukan sebelumnya.

```
def predict_next_words(seed_text, next_words):
  for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predict_x = model.predict(token_list)
    predicted = np.argmax(predict_x,axis=1)
    #predicted model.predict_classes(token list, verbose-8)
    output_word = ""
    for word, index in tokenizer.word_index.items():
      if index == predicted:
        output_word = word
        break
    seed_text += " " + output_word

  print(seed_text)
  return seed_text
```

6. Membuat Perintah untuk 100 Kata Selanjutnya

Setelah model selesai dilatih, dibuat sebuah fungsi bernama predict_next_words yang digunakan untuk memprediksi kata-kata berikutnya berdasarkan *seed text* (teks awal). Fungsi ini mengubah *seed text* menjadi urutan angka menggunakan tokenizer, lalu model memprediksi kata berikutnya sebanyak jumlah yang diinginkan (misalnya 100 kata). Proses ini dilakukan secara berulang, di mana setiap kata hasil prediksi ditambahkan ke *seed text* sebelumnya untuk menghasilkan teks lanjutan secara bertahap.

7. Output Artikel dengan Seed Text

[22]:
```
seed_text = '1THREEPIO'
next_words = 100

generated_text = predict_next_words(seed_text, next_words)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1THREEPIO to the love of the love of be show thee thee in i thee i thee i thee i thee i thee i thee i thee i thee i thee i
i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i
thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i thee i th
ee
```

[21]:
```
seed_text = 'why lovest thou that which not gladly'
generated_text = predict_next_words(seed_text, next_words)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
```

7. Output Artikel dengan Seed Text

Tahap terakhir adalah menguji kemampuan model dengan memberikan *seed text* tertentu, misalnya “why lovest thou that which not gladly”. Model kemudian menghasilkan teks lanjutan secara otomatis sebanyak 100 kata sesuai pola bahasa yang dipelajarinya dari karya Shakespeare. Hasil prediksi ini menunjukkan bagaimana model RNN dapat digunakan untuk menghasilkan teks bergaya Shakespeare secara otomatis, membuktikan bahwa model telah mempelajari struktur dan pola bahasa dari dataset yang digunakan.