Tugas Praktikum-14 Natural Language Processing (NLP)

Nama : Gavrilla Claudia

NIM : 21110004

Import Library Needed

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
```

Load Data

```
with open('/content/rihanna.txt', 'r', encoding='unicode_escape') as myfile:
    mytext = myfile.read()
```

```
mytext
```

```
'Ghost in the mirror\nI knew your face once, but now it\'s unclear\nAnd I can\'t feel my body now\nI\'m separate from here and now
A drug and a dream\nWe lost connection, oh come back to me\nSo I can feel alive again\nSoul and body try to mend It\'s pulling me
apart, this time\nEverything is never ending\nSlipped into a peril that\nI\'ll never understand\nThis feeling always gets away\nWi
shing I could hold on longer\nIt doesn\'t have to feel so strange\nTo be in love again, to be in love again, to be in love again G
host in the mirror\nI knew your face once, but now it\'s unclear\nAnd I can\'t feel my body now\nI\'m separate from here and now I
t\'s pulling me apart, this time\nEverything is never ending\nSlipped into a peril that\nI\'ll never understand\nThis feeling alwa
ys gets away\nWishing I could hold on longer\nIt doesn\'t have to feel so strange\nTo be in love again, to be in love again, to be
in love again Be in love again [Intro: Rihanna]\nBaby baby baby baby baby yeah\nUsed to l  '
```

Preprocessing

```
my_tokenizer = Tokenizer()
my_tokenizer.fit_on_texts([mytext])
total_words = len(my_tokenizer.word_index) + 1
```

```
my_tokenizer.word_index
```

```
 'tips': 692,
 'sixty': 693,
 'murderer': 694,
 'doing': 695,
 'girls': 696,
 'ugly': 697,
 're': 698,
 'cold': 699,
 'pain': 700,
 'bring': 701,
 'meet': 702,
 'glass': 703,
 'words': 704,
 'ray': 705,
 'speak': 706,
 'scream': 707,
 'follow': 708,
 'backs': 709,
 'mountains': 710,
 "fuckin'": 711,
 'vision': 712,
 'ecstasy': 713,
 'become': 714,
 'energy': 715,
 'rays': 716,
 'almost': 717,
 'save': 718
```

Interpretasi Ouput :

Dari output tersebut, dapat dilihat bahwa kata-kata dalam teks diubah menjadi angka-angka yang sesuai untuk diproses oleh komputer. Setiap kata memiliki nomor indeks yang unik. Misalnya, kata 'you' berindeks 1, kata 'i'berindeks 2, kata 'the' berindeks 3, kata 'me' berindeks 4,kata 'to' berindeks 5, dan seterusnya.

```
my_input_sequences = []
for line in mytext.split('\n'):
    # print(line)
    token_list = my_tokenizer.texts_to_sequences([line])[0]
    print(token_list)
    for i in range(1, len(token_list)):
        my_n_gram_sequence = token_list[:i+1]
        my_input_sequences.append(my_n_gram_sequence)
        # print(input_sequences)
```

```
[584, 11, 3, 345]
[2, 369, 18, 179, 238, 24, 32, 39, 661]
[7, 2, 57, 52, 9, 316, 32]
[12, 662, 118, 85, 7, 32, 8, 809, 7, 8, 300]
[36, 221, 974, 22, 28, 88, 5, 4]
[19, 2, 50, 52, 247, 259]
[447, 7, 316, 346, 5, 810, 39, 585, 4, 260, 33, 96]
[222, 26, 71, 663]
[664, 123, 8, 665, 16]
[148, 71, 301]
[33, 153, 133, 586, 83]
[317, 2, 154, 143, 15, 448]
[6, 587, 68, 5, 52, 19, 666]
[5, 20, 11, 10, 259, 5, 20, 11, 10, 259, 5, 20, 11, 10, 259, 584, 11, 3, 345]
[2, 369, 18, 179, 238, 24, 32, 39, 661]
[7, 2, 57, 52, 9, 316, 32]
[12, 662, 118, 85, 7, 32, 39, 585, 4, 260, 33, 96]
[222, 26, 71, 663]
[664, 123, 8, 665, 16]
[148, 71, 301]
[33, 153, 133, 586, 83]
[317, 2, 154, 143, 15, 448]
[6, 587, 68, 5, 52, 19, 666]
[5, 20, 11, 10, 259, 5, 20, 11, 10, 259, 5, 20, 11, 10, 259, 20, 11, 10, 259, 975, 155]
[31, 31, 31, 31, 31, 14]
[78, 5, 10, 1, 78, 5, 10, 1]
[78, 5, 10, 1, 78, 5, 10, 1]
[78, 5, 10, 1, 78, 5, 10, 1]
[78, 5, 10, 1, 78, 5, 10, 1, 370, 513, 155]
[213, 73, 213, 273]
[22, 2, 20, 214, 667, 302, 3, 44, 1, 303, 4]
[302, 3, 44, 1, 156, 4, 61, 12, 514, 347, 155, 274, 275]
[78, 5, 10, 1, 78, 5, 10, 1]
[78, 5, 10, 1, 78, 5, 10, 1]
[78, 5, 10, 1, 78, 5, 10, 1, 31, 515]
[78, 5, 10, 1, 78, 5, 10, 1, 370, 588, 274, 275]
[191, 3, 44, 161, 348, 128]
[2, 66, 1, 117, 976]
[2, 668, 47, 1, 1, 668, 47, 4]
[35, 371, 1, 214, 347, 155, 274, 275]
[78, 5, 10, 1, 78, 5, 10, 1]
[78, 5, 10, 1, 78, 5, 10, 1]
```

```
[78, 5, 10, 1, 78, 5, 10, 1, 31, 515]
[78, 5, 10, 1, 78, 5, 10, 1, 811, 155, 274, 275]
[57, 167]
[10, 4, 13, 1, 167, 31]
[57, 167, 516]
[143, 4, 13, 1, 167]
[57, 167, 31, 57, 167, 10, 4]
[156, 4, 13, 1]
[57, 167, 31, 81, 167, 10, 4, 32]
[81, 167, 31, 34]
[10, 4, 13, 1, 45]
[50, 167, 31]
[10, 4, 13, 1, 45]
[977, 11, 3, 589]
[978, 11, 8, 979, 812]
```

Interpretasi Ouput :

Dari output tersebut, dapat dilihat bahwa urutan kata - kata (sequence) dimulai dari [584, 11, 3, 345] karena pada dataframe tersebut kata - kata yang muncul di awal adalah "Ghost in the mirror" yang mana kata "ghost" memiliki indeks 584, kata "in" memiliki indeks 11, kata "the" memiliki indeks 3, dan kata "mirror" memilik indeks 345.

```
my_input_sequences = []
for line in mytext.split('\n'):
    # print(line)
    token_list = my_tokenizer.texts_to_sequences([line])[0]
    # print(token_list)
    for i in range(1, len(token_list)):
        my_n_gram_sequence = token_list[:i+1]
        print(my_n_gram_sequence)
        my_input_sequences.append(my_n_gram_sequence)
        # print(input_sequences)
```

```
Streaming output truncated to the last 5000 lines.
[12, 19, 70, 19, 70, 138]
[12, 19, 70, 19, 70, 138, 14]
[12, 19, 70, 19, 70, 138, 14, 14]
[12, 19, 70, 19, 70, 138, 14, 14, 14]
[12, 19]
[12, 19, 70]
[12, 19, 70, 87]
[12, 19, 70, 87, 70]
[12, 19, 70, 87, 70, 16]
[12, 19, 70, 87, 70, 16, 2]
[12, 19, 70, 87, 70, 16, 2, 2]
[12, 19, 70, 87, 70, 16, 2, 2, 2]
[12, 19]
[12, 19, 70]
[12, 19, 70, 19]
[12, 19, 70, 19, 70]
[12, 19, 70, 19, 70, 138]
[12, 19, 70, 19, 70, 138, 14]
[12, 19, 70, 19, 70, 138, 14, 14]
[12, 19, 70, 19, 70, 138, 14, 14, 14]
[12, 19]
[12, 19, 70]
[12, 19, 70, 87]
[12, 19, 70, 87, 70]
[19, 70]
[19, 70, 19]
[19, 70, 19, 70]
[19, 70, 19, 70, 19]
[19, 70, 19, 70, 19, 70]
[19, 70, 19, 70, 19, 70, 19]
[19, 70, 19, 70, 19, 70, 19, 70]
[19, 70, 19, 70, 19, 70, 19, 70, 42]
[19, 70, 19, 70, 19, 70, 19, 70, 42, 70]
[19, 70, 19, 70, 19, 70, 19, 70, 42, 70, 558]
[19, 70, 19, 70, 19, 70, 19, 70, 42, 70, 558, 42]
[19, 70, 19, 70, 19, 70, 19, 70, 42, 70, 558, 42, 245]
[88, 5]
[88, 5, 18]
[88, 5, 18, 1888]
[601, 3]
[601, 3, 1177]
[601, 3, 1177, 1889]
[601, 3, 1177, 1889, 172]
[601, 3, 1177, 1889, 172, 101]
[601, 3, 1177, 1889, 172, 101, 3]
[601, 3, 1177, 1889, 172, 101, 3, 1290]
[601, 3, 1177, 1889, 172, 101, 3, 1290, 88]
[601, 3, 1177, 1889, 172, 101, 3, 1290, 88, 5]
[601, 3, 1177, 1889, 172, 101, 3, 1290, 88, 5, 3]
[601, 3, 1177, 1889, 172, 101, 3, 1290, 88, 5, 3, 1890]
```

```
[2, 78]
[2, 78, 5]
[2, 78, 5, 125]
[2, 78, 5, 125, 9]
[2, 78, 5, 125, 9, 502]
[2, 78, 5, 125, 9, 502, 1290]
[2, 78, 5, 125, 9, 502, 1290, 13]
```

```
max_sequence_len = max([len(seq) for seq in my_input_sequences])
input_sequences = np.array(pad_sequences(my_input_sequences, maxlen=max_sequence_len, padding='pre'))
```

```
input_sequences
```

```
array([[  0,   0,   0, ...,   0, 584,  11],
       [  0,   0,   0, ..., 584,  11,   3],
       [  0,   0,   0, ...,  11,   3, 345],
       ...,
       [  0,   0,   0, ...,  36, 104,   3],
       [  0,   0,   0, ..., 104,   3, 183],
       [  0,   0,   0, ...,   3, 183, 299]], dtype=int32)
```

```
X = input_sequences[:, :-1]
y = input_sequences[:, -1]
```

```
X[2]
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0, 584,  11,   3], dtype=int32)
```

```
y[2]
```

```
345
```

```
X
```

```
array([[  0,   0,   0, ...,   0,   0, 584],
       [  0,   0,   0, ...,   0, 584,  11],
       [  0,   0,   0, ..., 584,  11,   3],
       ...,
       [  0,   0,   0, ...,   0,  36, 104],
       [  0,   0,   0, ...,  36, 104,   3],
       [  0,   0,   0, ..., 104,   3, 183]], dtype=int32)
```

```
y
```

```
array([ 11,   3, 345, ...,   3, 183, 299], dtype=int32)
```

```
# lakukan one hot encoding
y = np.array(tf.keras.utils.to_categorical(y, num_classes=total_words))
```

```
y
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
y[0]
```

```
array([0., 0., 0., ..., 0., 0., 0.], dtype=float32)
```

Define Models

```
model = tf.keras.models.Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(LSTM(150))
model.add(Dense(total_words, activation='softmax'))
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 embedding_1 (Embedding)     (None, 37, 100)           201800

 lstm_1 (LSTM)               (None, 150)               150600

 dense_1 (Dense)             (None, 2018)              304718

=================================================================
Total params: 657118 (2.51 MB)
Trainable params: 657118 (2.51 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
hist = model.fit(X, y, epochs=100, verbose=1)
```

```
Epoch 72/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4232 - accuracy: 0.8569
Epoch 73/100
822/822 [==============================] - 6s 7ms/step - loss: 0.4223 - accuracy: 0.8571
Epoch 74/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4217 - accuracy: 0.8573
Epoch 75/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4228 - accuracy: 0.8574
Epoch 76/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4219 - accuracy: 0.8574
Epoch 77/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4198 - accuracy: 0.8579
Epoch 78/100
822/822 [==============================] - 5s 7ms/step - loss: 0.4218 - accuracy: 0.8574
Epoch 79/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4223 - accuracy: 0.8574
Epoch 80/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4212 - accuracy: 0.8573
Epoch 81/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4204 - accuracy: 0.8584
Epoch 82/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4192 - accuracy: 0.8582
Epoch 83/100
822/822 [==============================] - 6s 7ms/step - loss: 0.4196 - accuracy: 0.8584
Epoch 84/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4216 - accuracy: 0.8561
Epoch 85/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4249 - accuracy: 0.8561
Epoch 86/100
822/822 [==============================] - 5s 7ms/step - loss: 0.4237 - accuracy: 0.8566
Epoch 87/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4209 - accuracy: 0.8588
Epoch 88/100
822/822 [==============================] - 6s 7ms/step - loss: 0.4175 - accuracy: 0.8578
Epoch 89/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4185 - accuracy: 0.8576
Epoch 90/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4195 - accuracy: 0.8573
Epoch 91/100
822/822 [==============================] - 6s 7ms/step - loss: 0.4198 - accuracy: 0.8571
Epoch 92/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4221 - accuracy: 0.8579
Epoch 93/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4221 - accuracy: 0.8585
Epoch 94/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4189 - accuracy: 0.8585
Epoch 95/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4203 - accuracy: 0.8565
Epoch 96/100
822/822 [==============================] - 5s 7ms/step - loss: 0.4179 - accuracy: 0.8581
Epoch 97/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4183 - accuracy: 0.8582
Epoch 98/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4214 - accuracy: 0.8569
Epoch 99/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4198 - accuracy: 0.8579
Epoch 100/100
822/822 [==============================] - 5s 6ms/step - loss: 0.4173 - accuracy: 0.8589
```

Interpretasi Ouput :

Dari output tersebut, dapat dilihat bahwa nilai loss: 0.4173 artinya model memiliki tingkat kehilangan yang relatif rendah. Dengan kata lain, model yang dibuat telah melakukan prediksi yang lebih baik atau lebih dekat dengan nilai target pada data training. Sementara nilai accuracy: 0.8589. Artinya, model yang dibuat memiliki tingkat keakuratan sekitar 85.89% dalam memprediksi target pada data training. Dengan kata lain, sekitar 85.89% dari prediksi yang dilakukan oleh model adalah benar berdasarkan data training.

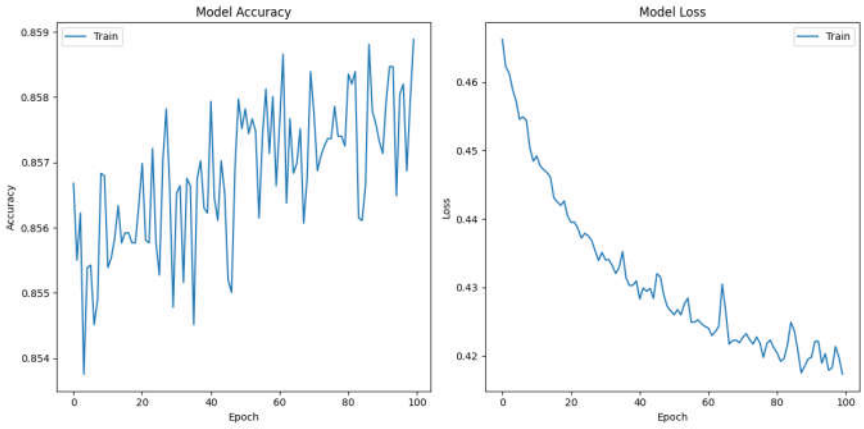
```
import matplotlib.pyplot as plt
```

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(hist.history['accuracy'], label='Train')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(hist.history['loss'], label='Train')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Interpretasi Ouput :

Dari output tersebut, dapat dilihat bahwa akurasi model memiliki tingkat keakuratan sekitar 85.89% dalam memprediksi target pada data training. Dengan kata lain, sekitar 85.89% dari prediksi yang dilakukan oleh model adalah benar berdasarkan data training. Sementara untuk model loss memiliki nilai sebesar 0.4173 atau 41.73% artinya model memiliki tingkat kehilangan yang relatif rendah. Dengan kata lain, model yang dibuat telah melakukan prediksi yang lebih baik atau lebih dekat dengan nilai target pada data training.

```
model.save("mymodel.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file v
  saving_api.save_model(
```

Make Prediction

```
model_loaded = load_model("mymodel.h5")
```

```
import numpy as np

input_text = "Rihanna"
predict_next_words = 5

for _ in range(predict_next_words):
    token_list = my_tokenizer.texts_to_sequences([input_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predictions = model_loaded.predict(token_list)[0]

    # Get indices of top predicted words
    top_indices = np.argsort(predictions)[-5:][::-1]  # Adjust 5 to the number of top words you want

    # Get words corresponding to the indices
    next_words = [word for word, index in my_tokenizer.word_index.items() if index in top_indices]

    # Print the list of next words along with their probabilities
    print("Input Text:", input_text)
    print("Next Words and Probabilities:")
    for word, index in zip(next_words, top_indices):
        probability = predictions[index]
        print(f"{word}: {probability:.4f}")

    # Choose the word with the highest probability as the next word
    output_word = my_tokenizer.index_word[top_indices[0]]

    input_text += " " + output_word

print(input_text)
```

```
 1/1 [==============================] - 0s 18ms/step
 Input Text: Rihanna
 Next Words and Probabilities:
 you: 0.9878
 just: 0.0032
 where: 0.0022
 girl: 0.0015
 good: 0.0009
 1/1 [==============================] - 0s 17ms/step
 Input Text: Rihanna where
 Next Words and Probabilities:
 you: 0.9971
 i: 0.0014
 more: 0.0006
 they: 0.0003
 touch: 0.0002
 1/1 [==============================] - 0s 18ms/step
 Input Text: Rihanna where you
 Next Words and Probabilities:
 come: 0.9946
 at: 0.0032
 please: 0.0003
 ya: 0.0003
 touch: 0.0002
 1/1 [==============================] - 0s 20ms/step
 Input Text: Rihanna where you at
 Next Words and Probabilities:
 you: 0.9800
 i: 0.0082
 yeah: 0.0018
 with: 0.0016
 where: 0.0014
 1/1 [==============================] - 0s 19ms/step
 Input Text: Rihanna where you at you
 Next Words and Probabilities:
 you: 0.9963
 don't: 0.0014
 let: 0.0007
 can't: 0.0002
 had: 0.0002
 Rihanna where you at you had
```

Interpretasi Ouput :

Dari output tersebut, dapat dilihat bahwa teks yang dimasukkan adalah kata "Rihanna". Kemudian model yang digunakan memprediksi kata-kata berikutnya dalam sebuah teks setelah kata "Rihanna" adalah kata "you" dengan probabilitas sebesar 0.9878.

```
input_text = "Don't stop the music"
predict_next_words = 10

for _ in range(predict_next_words):
    token_list = my_tokenizer.texts_to_sequences([input_text])[0]
    print(token_list)
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = np.argmax(model_loaded.predict(token_list), axis=-1)
    output_word = ""
    for word, index in my_tokenizer.word_index.items():
        if index == predicted:
            output_word = word
            break
    input_text += " " + output_word

print(input_text)
```

```
 [17, 67, 3, 53]
 1/1 [==============================] - 0s 26ms/step
 [17, 67, 3, 53, 53]
 1/1 [==============================] - 0s 26ms/step
 [17, 67, 3, 53, 53, 53]
 1/1 [==============================] - 0s 27ms/step
 [17, 67, 3, 53, 53, 53, 39]
 1/1 [==============================] - 0s 28ms/step
 [17, 67, 3, 53, 53, 53, 39, 252]
 1/1 [==============================] - 0s 26ms/step
 [17, 67, 3, 53, 53, 53, 39, 252, 193]
 1/1 [==============================] - 0s 26ms/step
 [17, 67, 3, 53, 53, 53, 39, 252, 193, 12]
 1/1 [==============================] - 0s 29ms/step
 [17, 67, 3, 53, 53, 53, 39, 252, 193, 12, 473]
 1/1 [==============================] - 0s 27ms/step
 [17, 67, 3, 53, 53, 53, 39, 252, 193, 12, 473, 9]
 1/1 [==============================] - 0s 40ms/step
 [17, 67, 3, 53, 53, 53, 39, 252, 193, 12, 473, 9, 44]
 1/1 [==============================] - 0s 26ms/step
 Don't stop the music music music it's getting late i'm making my way over
```

Save Model

```
model.save("mymodel.h5")
```

Load Model

```
model_loaded = load_model("mymodel.h5")


input_text = "Umbrella"
predict_next_words = 15

for _ in range(predict_next_words):
    token_list = my_tokenizer.texts_to_sequences([input_text])[0]
    print(token_list)
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = np.argmax(model_loaded.predict(token_list), axis=-1)
    output_word = ""
    for word, index in my_tokenizer.word_index.items():
        if index == predicted:
            output_word = word
            break
    input_text += " " + output_word

print(input_text)
```

```
 [307]
 1/1 [==============================] - 0s 26ms/step
 [307, 91]
 1/1 [==============================] - 0s 25ms/step
 [307, 91, 216]
 1/1 [==============================] - 0s 26ms/step
 [307, 91, 216, 412]
 1/1 [==============================] - 0s 29ms/step
 [307, 91, 216, 412, 169]
 1/1 [==============================] - 0s 35ms/step
 [307, 91, 216, 412, 169, 33]
 1/1 [==============================] - 0s 25ms/step
 [307, 91, 216, 412, 169, 33, 26]
 1/1 [==============================] - 0s 25ms/step
 [307, 91, 216, 412, 169, 33, 26, 297]
```

```
1/1 [==============================] - 0s 28ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12]
1/1 [==============================] - 0s 27ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12, 75]
1/1 [==============================] - 0s 26ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12, 75, 297]
1/1 [==============================] - 0s 28ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12, 75, 297, 11]
1/1 [==============================] - 0s 33ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12, 75, 297, 11, 10]
1/1 [==============================] - 0s 25ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12, 75, 297, 11, 10, 46]
1/1 [==============================] - 0s 36ms/step
[307, 91, 216, 412, 169, 33, 26, 297, 12, 75, 297, 11, 10, 46, 18]
1/1 [==============================] - 0s 28ms/step
Umbrella better forever well than this is stupid i'm not stupid in love you're your tiny
```