```
import numpy as np
from keras.models import Sequential
from keras.layers import SimpleRNN,Dense,Embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical


sentences=["I love programs","I love python","I hate school","Recurrent Neural Network are powerful"]


import pandas as pd
df=pd.read_csv("/content/earth.txt",sep=".")


tokenizer = Tokenizer()
tokenizer.fit_on_texts(df)
total_words=len(tokenizer.word_index)+1
print(total_words)
```

```
60
```

```
# Creating input sequences and their corresponding next words
input_sequences = []
for sentence in df:
    tokenized_sentence = tokenizer.texts_to_sequences([sentence])[0]
    for i in range(1, len(tokenized_sentence)):
        n_gram_sequence = tokenized_sentence[:i+1]
        input_sequences.append(n_gram_sequence)
input_sequences
```

```
[[17, 8],
 [17, 8, 4],
 [17, 8, 4, 18],
 [17, 8, 4, 18, 5],
 [17, 8, 4, 18, 5, 19],
 [17, 8, 4, 18, 5, 19, 20],
 [17, 8, 4, 18, 5, 19, 20, 21],
 [17, 8, 4, 18, 5, 19, 20, 21, 9],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12, 13],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12, 13, 23],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12, 13, 23, 2],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12, 13, 23, 2, 24],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12, 13, 23, 2, 24, 14],
 [17, 8, 4, 18, 5, 19, 20, 21, 9, 10, 2, 22, 11, 12, 13, 23, 2, 24, 14, 25],
 [17,
  8,
  4,
  18,
  5,
  19,
  20,
  21,
  9,
  10,
  2,
  22,
  11,
  12,
  13,
  23,
  2,
  24,
  14,
  25,
  26],
 [11, 8],
 [11, 8, 4],
 [11, 8, 4, 27],
 [11, 8, 4, 27, 6],
 [11, 8, 4, 27, 6, 28],
 [11, 8, 4, 27, 6, 28, 29],
 [11, 8, 4, 27, 6, 28, 29, 2],
 [11, 8, 4, 27, 6, 28, 29, 2, 30],
 [3, 7],
 [3, 7, 31],
 [3, 7, 31, 32],
 [3, 7, 31, 32, 33],
 [3, 7, 31, 32, 33, 15],
 [3, 7, 31, 32, 33, 15, 1],
 [3, 7, 31, 32, 33, 15, 1, 5],
```

```
        [3, 7, 31, 32, 33, 15, 1, 5, 34],
        [3, 7, 31, 32, 33, 15, 1, 5, 34, 35],
        [3, 7, 31, 32, 33, 15, 1, 5, 34, 35, 36],
```

```
# Padding sequences for consistent input size
max_sequence_length = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
```

```
input_sequences
```

```
array([[ 0,  0,  0, ...,  0, 17,  8],
       [ 0,  0,  0, ..., 17,  8,  4],
       [ 0,  0,  0, ...,  8,  4, 18],
       ...,
       [ 0,  0,  1, ..., 58, 12,  1],
       [ 0,  1,  3, ..., 12,  1, 59],
       [ 1,  3,  7, ...,  1, 59, 14]], dtype=int32)
```

```
# Creating input and output data
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = to_categorical(y, num_classes=total_words)
```

```
# Building a simple RNN model
model = Sequential()
model.add(Embedding(input_dim=total_words, output_dim=50, input_length=max_sequence_length-1))
model.add(SimpleRNN(100, return_sequences=True))
model.add(SimpleRNN(100))
model.add(Dense(total_words, activation='softmax'))
```

```
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"])
model.fit(X,y,epochs=50,verbose=2)
```

```
Epoch 1/50
3/3 - 2s - loss: 4.1321 - accuracy: 0.0118 - 2s/epoch - 729ms/step
Epoch 2/50
3/3 - 0s - loss: 3.9714 - accuracy: 0.0471 - 70ms/epoch - 23ms/step
Epoch 3/50
3/3 - 0s - loss: 3.7068 - accuracy: 0.2353 - 71ms/epoch - 24ms/step
Epoch 4/50
3/3 - 0s - loss: 3.5583 - accuracy: 0.3176 - 79ms/epoch - 26ms/step
Epoch 5/50
3/3 - 0s - loss: 3.4106 - accuracy: 0.4000 - 82ms/epoch - 27ms/step
Epoch 6/50
3/3 - 0s - loss: 3.2586 - accuracy: 0.4353 - 71ms/epoch - 24ms/step
Epoch 7/50
3/3 - 0s - loss: 3.1213 - accuracy: 0.4824 - 71ms/epoch - 24ms/step
Epoch 8/50
3/3 - 0s - loss: 2.9955 - accuracy: 0.4824 - 78ms/epoch - 26ms/step
Epoch 9/50
3/3 - 0s - loss: 2.8412 - accuracy: 0.4824 - 72ms/epoch - 24ms/step
Epoch 10/50
3/3 - 0s - loss: 2.7195 - accuracy: 0.5294 - 84ms/epoch - 28ms/step
Epoch 11/50
3/3 - 0s - loss: 2.5826 - accuracy: 0.5647 - 71ms/epoch - 24ms/step
Epoch 12/50
3/3 - 0s - loss: 2.4463 - accuracy: 0.6118 - 75ms/epoch - 25ms/step
Epoch 13/50
3/3 - 0s - loss: 2.3151 - accuracy: 0.6471 - 68ms/epoch - 23ms/step
Epoch 14/50
3/3 - 0s - loss: 2.1863 - accuracy: 0.7059 - 73ms/epoch - 24ms/step
Epoch 15/50
3/3 - 0s - loss: 2.0585 - accuracy: 0.7294 - 86ms/epoch - 29ms/step
Epoch 16/50
3/3 - 0s - loss: 1.9316 - accuracy: 0.7412 - 71ms/epoch - 24ms/step
Epoch 17/50
3/3 - 0s - loss: 1.8101 - accuracy: 0.7882 - 68ms/epoch - 23ms/step
Epoch 18/50
3/3 - 0s - loss: 1.7009 - accuracy: 0.8000 - 89ms/epoch - 30ms/step
Epoch 19/50
3/3 - 0s - loss: 1.5827 - accuracy: 0.8118 - 71ms/epoch - 24ms/step
Epoch 20/50
3/3 - 0s - loss: 1.4932 - accuracy: 0.7882 - 76ms/epoch - 25ms/step
Epoch 21/50
3/3 - 0s - loss: 1.3867 - accuracy: 0.8235 - 73ms/epoch - 24ms/step
Epoch 22/50
3/3 - 0s - loss: 1.2995 - accuracy: 0.8235 - 70ms/epoch - 23ms/step
Epoch 23/50
3/3 - 0s - loss: 1.2166 - accuracy: 0.8235 - 83ms/epoch - 28ms/step
Epoch 24/50
3/3 - 0s - loss: 1.1328 - accuracy: 0.8235 - 71ms/epoch - 24ms/step
Epoch 25/50
3/3 - 0s - loss: 1.0603 - accuracy: 0.8235 - 72ms/epoch - 24ms/step
Epoch 26/50
3/3 - 0s - loss: 0.9954 - accuracy: 0.8353 - 71ms/epoch - 24ms/step
Epoch 27/50
```

```
3/3 - 0s - loss: 0.9237 - accuracy: 0.8706 - 71ms/epoch - 24ms/step
Epoch 28/50
3/3 - 0s - loss: 0.8662 - accuracy: 0.8588 - 68ms/epoch - 23ms/step
Epoch 29/50
3/3 - 0s - loss: 0.8155 - accuracy: 0.8588 - 71ms/epoch - 24ms/step
```

```
# Generating text using the trained model
seed_text = input("Enter the starting word: ")
next_words = int(input("Enter how many words to predict: "))

for _ in range(next_words):
    tokenized_seed = tokenizer.texts_to_sequences([seed_text])[0]
    tokenized_seed = pad_sequences([tokenized_seed], maxlen=max_sequence_length-1, padding='pre')
    predicted_word_index = np.argmax(model.predict(tokenized_seed), axis=-1)
    predicted_word = tokenizer.index_word[predicted_word_index[0]]
    seed_text += " " + predicted_word

print(seed_text)
```

```
Enter the starting word: i
Enter how many words to predict: 4
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
i has has a composition
```