```
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"]
```

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

```
12
```

```
# Creating input sequences and their corresponding next words
input_sequences = []
for sentence in sentences:
    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
```

```
[[1, 2],
 [1, 2, 3],
 [1, 2],
 [1, 2, 4],
 [1, 5],
 [1, 5, 6],
 [7, 8],
 [7, 8, 9],
 [7, 8, 9, 10],
 [7, 8, 9, 10, 11]]
```

```
# 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,  1,  2],
       [ 0,  0,  1,  2,  3],
       [ 0,  0,  0,  1,  2],
       [ 0,  0,  1,  2,  4],
       [ 0,  0,  0,  1,  5],
       [ 0,  0,  1,  5,  6],
       [ 0,  0,  0,  7,  8],
       [ 0,  0,  7,  8,  9],
       [ 0,  7,  8,  9, 10],
       [ 7,  8,  9, 10, 11]], 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
1/1 - 2s - loss: 2.4600 - accuracy: 0.0000e+00 - 2s/epoch - 2s/step
Epoch 2/50
1/1 - 0s - loss: 2.3700 - accuracy: 0.3000 - 14ms/epoch - 14ms/step
Epoch 3/50
1/1 - 0s - loss: 2.2822 - accuracy: 0.3000 - 11ms/epoch - 11ms/step
Epoch 4/50
1/1 - 0s - loss: 2.1953 - accuracy: 0.3000 - 12ms/epoch - 12ms/step
Epoch 5/50
1/1 - 0s - loss: 2.1100 - accuracy: 0.3000 - 10ms/epoch - 10ms/step
Epoch 6/50
```

```
1/1 - 0s - loss: 2.0290 - accuracy: 0.3000 - 11ms/epoch - 11ms/step
Epoch 7/50
1/1 - 0s - loss: 1.9547 - accuracy: 0.3000 - 11ms/epoch - 11ms/step
Epoch 8/50
1/1 - 0s - loss: 1.8861 - accuracy: 0.3000 - 10ms/epoch - 10ms/step
Epoch 9/50
1/1 - 0s - loss: 1.8175 - accuracy: 0.3000 - 12ms/epoch - 12ms/step
Epoch 10/50
1/1 - 0s - loss: 1.7438 - accuracy: 0.4000 - 11ms/epoch - 11ms/step
Epoch 11/50
1/1 - 0s - loss: 1.6659 - accuracy: 0.4000 - 11ms/epoch - 11ms/step
Epoch 12/50
1/1 - 0s - loss: 1.5889 - accuracy: 0.5000 - 10ms/epoch - 10ms/step
Epoch 13/50
1/1 - 0s - loss: 1.5167 - accuracy: 0.5000 - 11ms/epoch - 11ms/step
Epoch 14/50
1/1 - 0s - loss: 1.4494 - accuracy: 0.5000 - 12ms/epoch - 12ms/step
Epoch 15/50
1/1 - 0s - loss: 1.3844 - accuracy: 0.6000 - 15ms/epoch - 15ms/step
Epoch 16/50
1/1 - 0s - loss: 1.3200 - accuracy: 0.7000 - 12ms/epoch - 12ms/step
Epoch 17/50
1/1 - 0s - loss: 1.2576 - accuracy: 0.7000 - 12ms/epoch - 12ms/step
Epoch 18/50
1/1 - 0s - loss: 1.2000 - accuracy: 0.7000 - 10ms/epoch - 10ms/step
Epoch 19/50
1/1 - 0s - loss: 1.1479 - accuracy: 0.7000 - 10ms/epoch - 10ms/step
Epoch 20/50
1/1 - 0s - loss: 1.0990 - accuracy: 0.7000 - 11ms/epoch - 11ms/step
Epoch 21/50
1/1 - 0s - loss: 1.0504 - accuracy: 0.7000 - 11ms/epoch - 11ms/step
Epoch 22/50
1/1 - 0s - loss: 1.0025 - accuracy: 0.7000 - 13ms/epoch - 13ms/step
Epoch 23/50
1/1 - 0s - loss: 0.9576 - accuracy: 0.7000 - 11ms/epoch - 11ms/step
Epoch 24/50
1/1 - 0s - loss: 0.9165 - accuracy: 0.7000 - 12ms/epoch - 12ms/step
Epoch 25/50
1/1 - 0s - loss: 0.8771 - accuracy: 0.7000 - 12ms/epoch - 12ms/step
Epoch 26/50
1/1 - 0s - loss: 0.8381 - accuracy: 0.8000 - 12ms/epoch - 12ms/step
Epoch 27/50
1/1 - 0s - loss: 0.8003 - accuracy: 0.8000 - 11ms/epoch - 11ms/step
Epoch 28/50
1/1 - 0s - loss: 0.7651 - accuracy: 0.8000 - 11ms/epoch - 11ms/step
Epoch 29/50
1/1 - 0s - loss: 0.7333 - accuracy: 0.8000 - 10ms/epoch - 10ms/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: 11
1/1 [==============================] - 0s 254ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
I love programs neural programs powerful powerful powerful powerful powerful powerful powerful
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
---------------------------------------------------------------------------
MessageError                              Traceback (most recent call last)
<ipython-input-12-d5df0069828e> in <cell line: 2>()
      1 from google.colab import drive
----> 2 drive.mount('/content/drive')

                              3 frames
/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in read_reply_from_input(message_id, timeout_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
    104       return reply.get('data', None)
    105 

MessageError: Error: credential propagation was unsuccessful
```