A Recurrent Neural Network (RNN) is a neural network designed for sequential data.

Unlike N-gram:

It does not store fixed word combinations.

It maintains a hidden state (memory).

It learns patterns across sequences.

RNN predicts the next word based on previous words using learned weights instead of counting probabilities.

This allows better generalization than N-gram models.

```
# ==========================================
# RNN BASED TEXT GENERATION (WORD LEVEL)
# ==========================================

import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
import random

# -------------------------
# 1. Load Dataset
# -------------------------

text = """
artificial intelligence is transforming modern society
it is used in healthcare finance education and transportation
machine learning allows systems to improve automatically with experience
data plays a critical role in training intelligent systems
large datasets help models learn complex patterns
deep learning uses multi layer neural networks
neural networks are inspired by biological neurons
each neuron processes input and produces an output
training a neural network requires optimization techniques
gradient descent minimizes the loss function
"""

# -------------------------
# 2. Tokenization
# -------------------------

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1

# Convert text to sequence
input_sequences = []

for line in text.split("\n"):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# Pad sequences
max_seq_len = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_seq_len, padding='pre')

# Split into X and y
X = input_sequences[:, :-1]
y = input_sequences[:, -1]

# -------------------------
# 3. Build RNN Model
# -------------------------

model = Sequential([
    Embedding(total_words, 64, input_length=max_seq_len-1),
    SimpleRNN(128),
    Dense(total_words, activation='softmax')
])

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

model.summary()

# -------------------------
# 4. Train Model
# -------------------------

model.fit(X, y, epochs=20, verbose=1)

# -------------------------
# 5. Text Generation
# -------------------------

def generate_text(seed_text, next_words=20):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_seq_len-1, padding='pre')

        predicted = np.argmax(model.predict(token_list, verbose=0), axis=-1)

        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
            seed_text += " " + output_word

        return seed_text

    print("\nGenerated Text:\n")
    print(generate_text("artificial intelligence", 20))
```

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_1 (Embedding)         │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ simple_rnn_1 (SimpleRNN)        │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ ?                      │   0 (unbuilt) │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 460ms/step - accuracy: 0.0000e+00 - loss: 4.1847
Epoch 2/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1474 - loss: 4.1079
Epoch 3/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1394 - loss: 4.0628
Epoch 4/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1977 - loss: 4.0206
Epoch 5/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.2211 - loss: 3.9631
Epoch 6/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.2714 - loss: 3.9155
Epoch 7/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.2365 - loss: 3.8563
Epoch 8/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.1472 - loss: 3.8058
Epoch 9/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.1550 - loss: 3.7624
Epoch 10/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.1549 - loss: 3.7132
Epoch 11/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.1513 - loss: 3.6781
Epoch 12/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.1162 - loss: 3.6729
Epoch 13/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.1356 - loss: 3.6231
Epoch 14/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.2443 - loss: 3.4965
Epoch 15/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.3450 - loss: 3.4268
Epoch 16/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3798 - loss: 3.3523
Epoch 17/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.4344 - loss: 3.2340
Epoch 18/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.3837 - loss: 3.1992
Epoch 19/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.3451 - loss: 3.1030
Epoch 20/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.3061 - loss: 3.0866

Generated Text:

artificial intelligence is neural networks networks networks by biological training training intelligent experience experience transportation transportation experience
```

Limitations of Basic RNN

❌ Vanishing gradient problem

❌ Cannot handle long-term dependencies well

❌ Training is slow for long sequences

❌ Performance degrades for very long text

Because of this, we later improve it using LSTM / GRU.