🔹 Introduction (Medium Summary)

LSTM (Long Short-Term Memory) is an improved version of RNN.

It contains:

Input gate

Forget gate

Output gate

These gates help the model decide:

What to remember

What to forget

Because of this, LSTM handles long-term dependencies better than SimpleRNN.

```
# ==========================================
# LSTM 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, LSTM, Dense

# -------------------------
# 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

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)

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

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

# -------------------------
# 3. Build LSTM Model
# -------------------------

model = Sequential([
    Embedding(total_words, 64, input_length=max_seq_len-1),
    LSTM(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=30, verbose=1)

# -------------------------
# 5. Generate Text
# -------------------------

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_4"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_4 (Embedding)         │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_4 (LSTM)                   │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_4 (Dense)                 │ ?                      │   0 (unbuilt) │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.0000e+00 - loss: 4.1917
Epoch 2/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0426 - loss: 4.1825
Epoch 3/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0116 - loss: 4.1756
Epoch 4/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.0310 - loss: 4.1697
Epoch 5/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.0310 - loss: 4.1636
Epoch 6/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0310 - loss: 4.1560
Epoch 7/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0310 - loss: 4.1461
Epoch 8/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0464 - loss: 4.1408
Epoch 9/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0620 - loss: 4.1283
Epoch 10/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0542 - loss: 4.1184
Epoch 11/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0464 - loss: 4.1009
Epoch 12/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0892 - loss: 4.0816
Epoch 13/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0698 - loss: 4.0523
Epoch 14/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0698 - loss: 4.0239
Epoch 15/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0542 - loss: 3.9764
Epoch 16/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0582 - loss: 3.9213
Epoch 17/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0387 - loss: 3.9033
Epoch 18/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.0620 - loss: 3.9240
Epoch 19/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0698 - loss: 3.9288
Epoch 20/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0736 - loss: 3.8779
Epoch 21/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0774 - loss: 3.8715
Epoch 22/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0930 - loss: 3.8413
Epoch 23/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0774 - loss: 3.8198
Epoch 24/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1319 - loss: 3.8071
Epoch 25/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.1433 - loss: 3.7949
Epoch 26/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1162 - loss: 3.7824
Epoch 27/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1046 - loss: 3.7792
Epoch 28/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1007 - loss: 3.7688
Epoch 29/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.1397 - loss: 3.7037
Epoch 30/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.1434 - loss: 3.6688

Generated Text:

artificial intelligence is is is is is in healthcare with with with experience experience experience with experience experience experience with experience experience
```

🔹 Limitations of LSTM

❌ Slower than SimpleRNN

❌ Requires more computation

❌ Still sequential (not parallel like Transformers)

❌ Needs larger data for best performance

Next improvement after this → Transformers.