Q6. Simple RNN (single sequence, character level)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

# 1. Mapping
chars = ['h','e','l','o']
char_to_idx = {c:i for i,c in enumerate(chars)}
idx_to_char = {i:c for c,i in char_to_idx.items()}

# 2. Data (hello → prediction)
input_seq  = [0,1,2,2]    # h e l l
target_seq = [1,2,2,3]    # e l l o
X = np.array(input_seq).reshape(1,-1)
y = np.array(target_seq).reshape(1,-1)

# 3. Model
vocab_size = len(chars)
model = Sequential([
    Embedding(vocab_size, 8, input_length=X.shape[1]),
    SimpleRNN(8, return_sequences=True),
    Dense(vocab_size, activation='softmax')
])
model.compile('adam', 'sparse_categorical_crossentropy')

# 4. Train & Predict
model.fit(X, y, epochs=100, verbose=0)
pred = model.predict(np.array([[char_to_idx['h']]]), verbose=0)
idx = np.argmax(pred[0,0])
print(f"After 'h' → '{idx_to_char[idx]}'")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
After 'h' → 'e'
```

Q7. RNN with Multiple Words (char-level on sentence)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, SimpleRNN, Dense

# 1. Text & Mapping
txt = "hello how are you"
chars = sorted(set(txt))
c2i = {c:i for i,c in enumerate(chars)}
i2c = {i:c for c,i in c2i.items()}

# 2. Sequences
seq_len = 10
inputs, targets = [], []
for i in range(len(txt)-seq_len):
    seq_in  = txt[i:i+seq_len]
    seq_out = txt[i+1:i+seq_len+1]
    inputs.append([c2i[c] for c in seq_in])
    targets.append([c2i[c] for c in seq_out])
X = np.array(inputs)
y = np.array(targets)

# 3. Model
vocab = len(chars)
inp = Input((seq_len,))
x = Embedding(vocab,16)(inp)
x = SimpleRNN(64, return_sequences=True)(x)
out = Dense(vocab, activation='softmax')(x)
model = Model(inp, out)
model.compile('adam', 'sparse_categorical_crossentropy')

# 4. Train & Generate
model.fit(X, y, epochs=200, verbose=0)

def gen(txt_in, length=50):
    seq = [c2i[c] for c in txt_in]
    for _ in range(length):
        pad = np.array(seq[-seq_len:]).reshape(1,-1)
        p = model.predict(pad, verbose=0)[0,-1]
        seq.append(np.argmax(p))
    return ''.join(i2c[i] for i in seq)

print(gen("hello how "))
```

```
hello how are you yoe yo hyo are you yoe yo hyo are you yoe
```

Q8. Advanced RNN (stacked + bidirectional)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, SimpleRNN, Dense

# 1. Training Text & Character Mapping

training_text = "hello how are you doing today? this is an advanced rnn example."
chars = sorted(set(training_text))
char_to_idx = {ch: i for i, ch in enumerate(chars)}
idx_to_char = {i: ch for ch, i in char_to_idx.items()}
vocab_size = len(chars)

# 2. Prepare Sequence Data (windows)

sequence_length = 10
step = 1
input_sequences = []
target_sequences = []

for i in range(0, len(training_text) - sequence_length, step):
    input_chunk = training_text[i : i + sequence_length]
    target_chunk = training_text[i + 1 : i + sequence_length + 1]
    input_sequences.append([char_to_idx[ch] for ch in input_chunk])
    target_sequences.append([char_to_idx[ch] for ch in target_chunk])

X = np.array(input_sequences)              # shape: (num_samples, sequence_length)
y = np.array(target_sequences)             # shape: (num_samples, sequence_length)

# 3. Build the Model

embedding_dim = 32
rnn_units_1   = 64
rnn_units_2   = 64

model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=sequence_length),
    Bidirectional(SimpleRNN(units=rnn_units_1, return_sequences=True)),
    SimpleRNN(units=rnn_units_2, return_sequences=True),
    Dense(vocab_size, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy'
)

model.summary()

# 4. Train the Model
```

```
epochs = 100
batch_size = 128

model.fit(
    X,
    y,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.1,
    verbose=2
)

# 5. Sequence Generation Function

def generate_sequence(start_text, gen_length=100):

    # convert seed text to indices
    generated_indices = [char_to_idx.get(ch, 0) for ch in start_text]

    for _ in range(gen_length):
        # prepare last sequence_length inputs
        seq = generated_indices[-sequence_length:]
        if len(seq) < sequence_length:
            # pad with zeros on the left if needed
            seq = [0] * (sequence_length - len(seq)) + seq
        inp = np.array(seq).reshape(1, sequence_length)

        # predict next char distribution
        preds = model.predict(inp, verbose=0)
        next_idx = np.argmax(preds[0, -1])
        generated_indices.append(next_idx)

    # convert indices back to characters
    return ''.join(idx_to_char[i] for i in generated_indices)


# 6. Run Generation & Print

seed = "hello how "
generated = generate_sequence(seed, gen_length=200)
print("\n📝 Generated text:\n" + generated)
```

```
📝 Generated text:
hello how are you doing today? this is an advanced rnn exdoin et doing today? this is an advanced rnn exdoin et doing today? this is an advanced rnn exdoin et doing today? this is
```

Q9. RNN with Other Words (word-level)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

# 1. Word mapping
words = ["hello","how","are","you"]
w2i = {w:i for i,w in enumerate(words)}
i2w = {i:w for w,i in w2i.items()}

# 2. Sequence: hello how are → how are you
in_seq  = [w2i[w] for w in ["hello","how","are"]]
out_seq = [w2i[w] for w in ["how","are","you"]]
X = np.array(in_seq).reshape(1,-1)
y = np.array(out_seq).reshape(1,-1)

# 3. Model
model = Sequential([
    Embedding(len(words), 8, input_length=3),
    SimpleRNN(16, return_sequences=True),
    Dense(len(words), activation='softmax')
])
model.compile('adam','sparse_categorical_crossentropy')
model.fit(X, y, epochs=200, verbose=0)

# 4. Predict
pred = model.predict(X, verbose=0)
print([[i2w[np.argmax(step)] for step in seq] for seq in pred])
```

```
warnings.warn(
[['how', 'are', 'you']]
```

Q10. RNN with Paragraph (longer text)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

para = ("deep learning enables models to learn hierarchical "
        "representations of data for tasks like translation and vision")
chars = sorted(set(para))
c2i = {c:i for i,c in enumerate(chars)}
i2c = {i:c for c,i in c2i.items()}

L, step = 40, 3
X, y = [], []
for i in range(0, len(para)-L, step):
    X.append([c2i[c] for c in para[i:i+L]])
    y.append([c2i[c] for c in para[i+1:i+L+1]])
X, y = np.array(X), np.array(y)

model = Sequential([
    Embedding(len(chars), 32, input_length=L),
    SimpleRNN(128, return_sequences=True),
    Dense(len(chars), activation='softmax')
])
model.compile('adam', 'sparse_categorical_crossentropy')
model.fit(X, y, epochs=50, batch_size=64, verbose=2)

def gen(seed, length=100):
    seq = [c2i.get(c,0) for c in seed]
    for _ in range(length):
        window = seq[-L:]
        if len(window) < L:
            window = [0]*(L - len(window)) + window
        inp = np.array(window).reshape(1, L)
        p = model.predict(inp, verbose=0)
        seq.append(int(np.argmax(p[0, -1])))
    return ''.join(i2c[i] for i in seq)

print(gen(para[:10], 200))
```

```
deep learnaltctltaonsascot taorsl t tasasn fiaetart o taarsrntae sntiol tatnsenr taeeseto hesra chieat  eoretan snlias or deta for tarks le eatrsns  tiorsaof data for tasks like
```

Q11. LSTM (char-level single)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# data prep (from “hello” example)
text = "hello"
chars = sorted(set(text))
char_to_idx = {c:i for i,c in enumerate(chars)}
idx_to_char = {i:c for c,i in char_to_idx.items()}

input_seq  = [char_to_idx[c] for c in text[:-1]]
target_seq = [char_to_idx[c] for c in text[1:]]
X = np.array(input_seq).reshape(1, -1)
y = np.array(target_seq).reshape(1, -1)

vocab_size = len(chars)

# model
model = Sequential([
    Embedding(vocab_size, 8, input_length=X.shape[1]),
    LSTM(8, return_sequences=True),
    Dense(vocab_size, activation='softmax')
])
model.compile('adam', 'sparse_categorical_crossentropy')

# train & predict
model.fit(X, y, epochs=100, verbose=0)
pred = model.predict(np.array([[char_to_idx['h']]]), verbose=0)
print(idx_to_char[np.argmax(pred[0,0])])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
l
```

Q12. LSTM Sequence Prediction (numeric)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# synthetic Fibonacci-style data
data = list(range(1, 21))
X, y = [], []
for i in range(len(data) - 2):
    X.append(data[i:i+2])
    y.append(data[i+2])
X = np.array(X).reshape(-1, 2, 1)
y = np.array(y)

# model
model = Sequential([
    LSTM(16, input_shape=(2,1)),
    Dense(1)
])
model.compile('adam', 'mse')

# train & predict
model.fit(X, y, epochs=200, verbose=0)
last = np.array(data[-2:]).reshape(1, 2, 1)
print("Next num:", model.predict(last)[0,0])
```

```
1/1                         0s 194ms/step
Next num: 6.633176
```

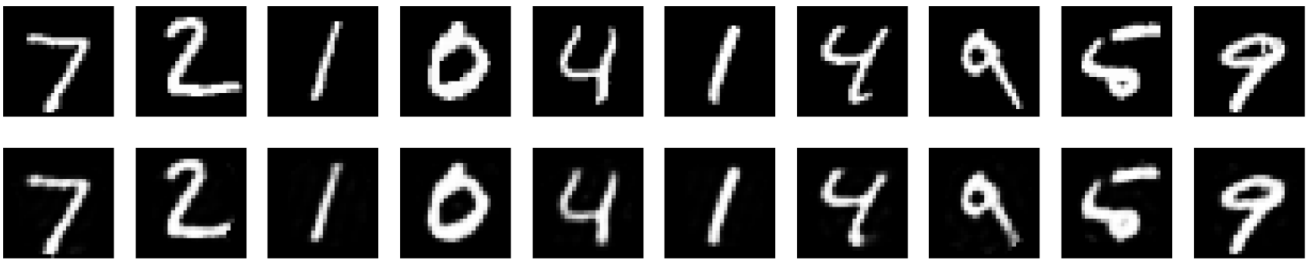
Q13. Multilayer LSTM for Sequence Prediction

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# reuse Q12 data
data = list(range(1, 21))
X, y = [], []
for i in range(len(data) - 2):
    X.append(data[i:i+2])
    y.append(data[i+2])
X = np.array(X).reshape(-1, 2, 1)
y = np.array(y)
last = np.array(data[-2:]).reshape(1, 2, 1)

# model
model = Sequential([
    LSTM(32, return_sequences=True, input_shape=(2,1)),
    LSTM(16),
    Dense(1)
])
model.compile('adam', 'mse')

# train & predict
model.fit(X, y, epochs=150, verbose=0)
print("Next num:", model.predict(last)[0,0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step
Next num: 6.299425
```

Q14. Auto-Encoder (dense on MNIST)

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.datasets import mnist

(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.reshape(-1, 784) / 255.
x_test = x_test.reshape(-1, 784) / 255.

inp = Input((784,))
encoded = Dense(64, activation='relu')(inp)
decoded = Dense(784, activation='sigmoid')(encoded)
auto = Model(inp, decoded)
auto.compile('adam', 'mse')

auto.fit(x_train, x_train, epochs=20, batch_size=256, validation_split=0.1, verbose=2)

# Predict
decoded_imgs = auto.predict(x_test)

# Display original and reconstruction
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
    plt.axis("off")

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    plt.axis("off")

plt.show()
```



Q15. Advanced Auto-Encoder (convolutional)

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.datasets import mnist

(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1) / 255.
x_test = x_test.reshape(-1, 28, 28, 1) / 255.

inp = Input((28, 28, 1))
x = Conv2D(32, 3, activation='relu', padding='same')(inp)
x = MaxPooling2D(2, padding='same')(x)
x = Conv2D(16, 3, activation='relu', padding='same')(x)
encoded = MaxPooling2D(2, padding='same')(x)

x = Conv2D(16, 3, activation='relu', padding='same')(encoded)
x = UpSampling2D(2)(x)
x = Conv2D(32, 3, activation='relu', padding='same')(x)
x = UpSampling2D(2)(x)
decoded = Conv2D(1, 3, activation='sigmoid', padding='same')(x)

auto = Model(inp, decoded)
auto.compile('adam', 'binary_crossentropy')
auto.fit(x_train, x_train, epochs=20, batch_size=128, validation_split=0.1, verbose=2)

# Predict
decoded_imgs = auto.predict(x_test)

# Display
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.show()
```