EXP No: 9 :- Build a Recurrent Neural Network

Aim:-

To implement a simple Recurrent Neural Network for sequential data prediction and analyze its performance.

Objectives:-

1. Understand the working of RNNs for sequential data
2. Train an RNN model on a time - series dataset.
3. Compare predicted and actual values to evalua-te performance

Algorithm:-

1. Data preprocessing:- Normalize the datasets and split into training and testing sets.
2. Model Design:- Define and RNN with input, hidden, and output layers.
3. Training: Feed sequences into the RNN, Comp-ute loss, and update weights using Back-propagation Through Time
4. Prediction:- Use the trained RNN to predict future values.
5. Evaluation:- Measure performance using metrics like MSE or ~~RSI~~ RMSE.

Pseudo Code:-

```
Load dataset
Normalize data
split dataset into train and test

Initialize RNN model
for each epoch:
    for each batch in training data:
        predict output
        Compute loss
        Backpropagate loss through time
        update weights.
predict on test data
Denormalize predictions.
Calculate evaluation metrices (MSE/RMSE)
```

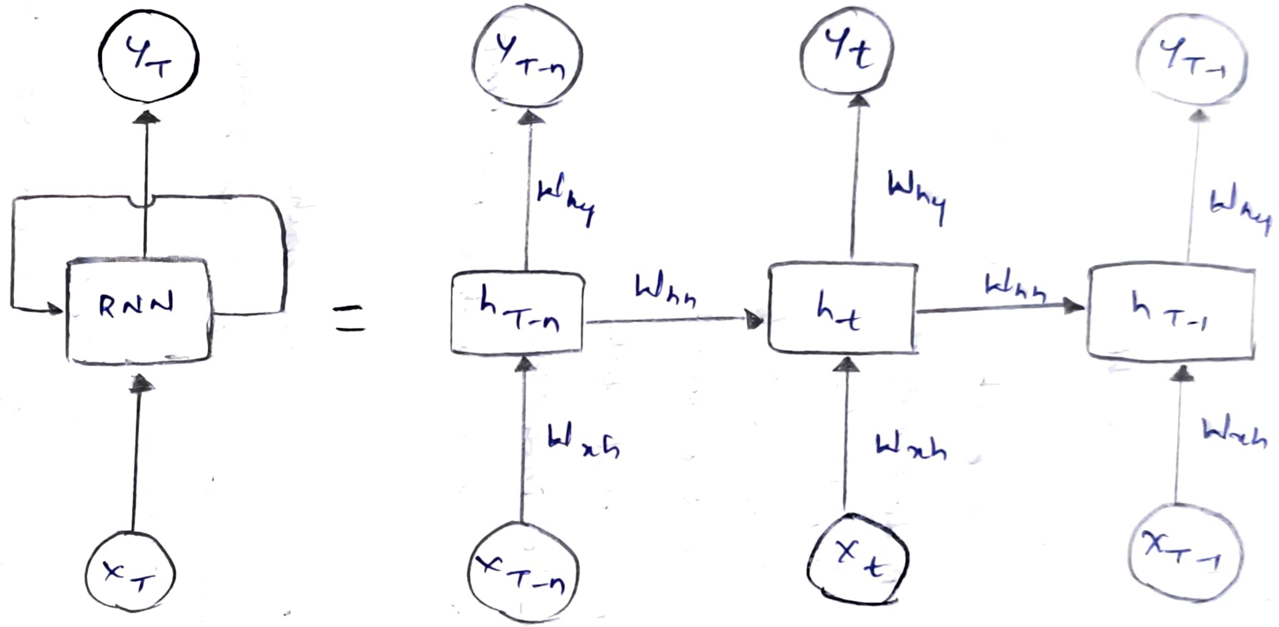
Observation:-

- Loss decreases gradually with training, but may plateau faster than LSTM
- RNN captures temporal patterns, but struggles with long-term dependencies.
- predictions follow general trends but may miss sharp fluctuations.

Conclusion:-

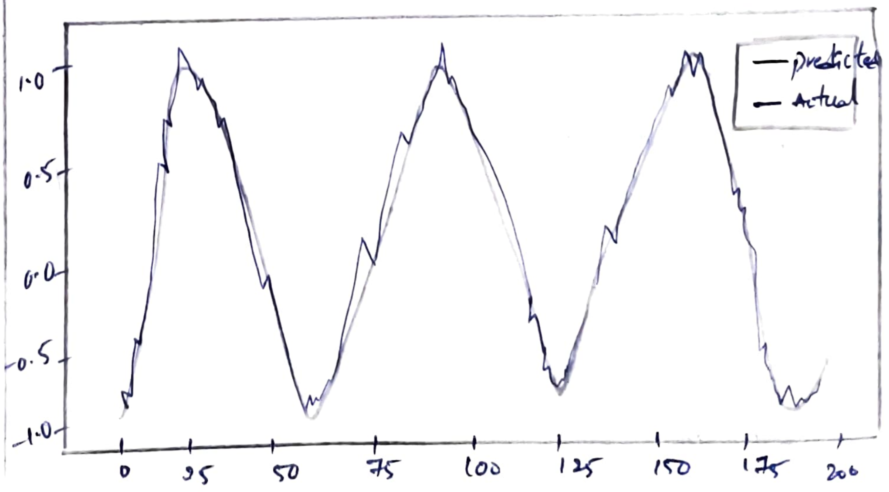
- RNNs are useful for sequential and time-series prediction but have limitations in learning long-term dependencies.

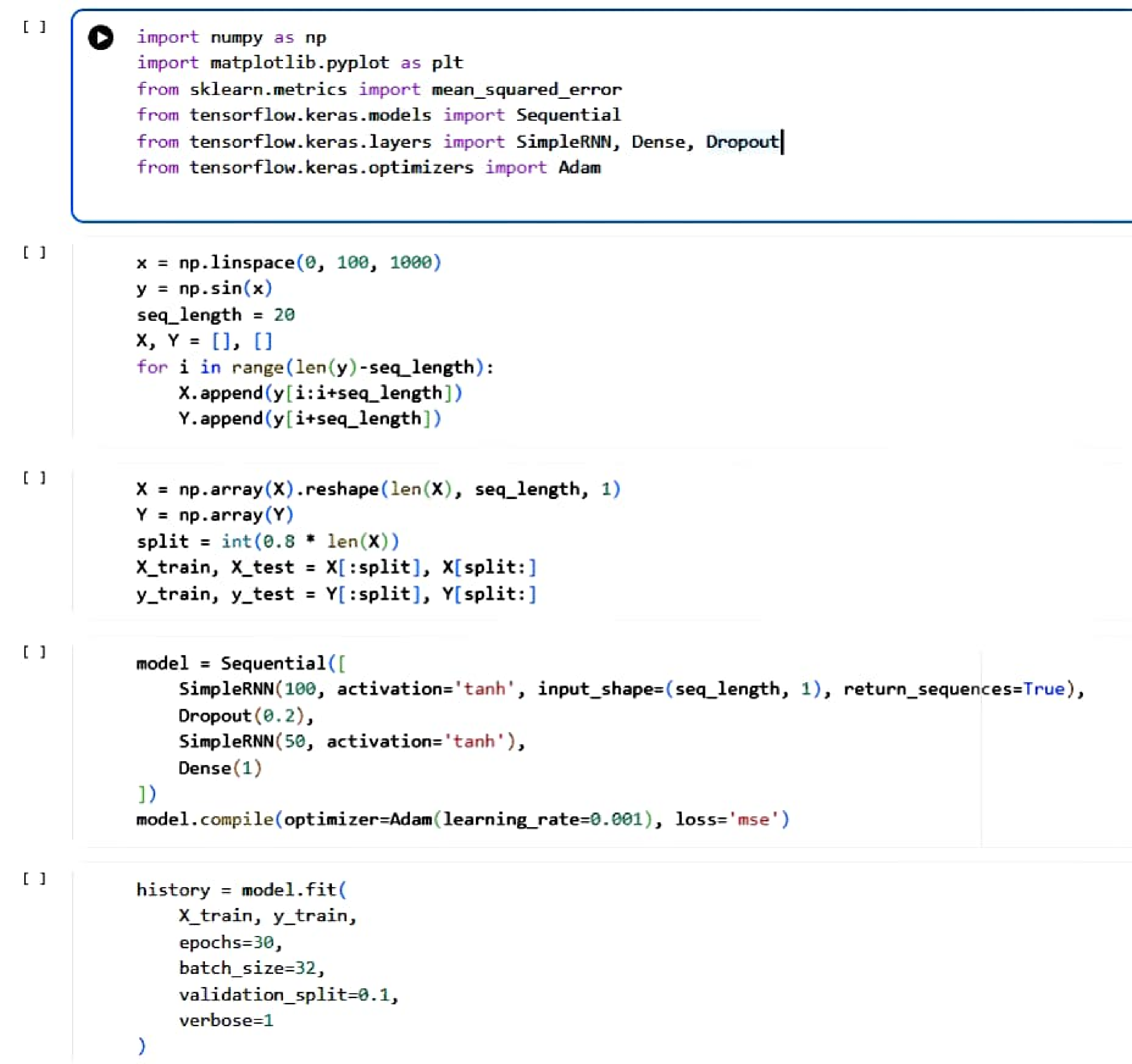
A SIMPLE ARCHITECTURE OF RNN MODEL



output:

Epoch	step_loss	val_loss
1	0.5478	0.0334
2	0.0259	0.0181
3	0.0166	0.0196
4	0.0157	0.0172
5	0.0150	0.0161
6	0.0150	0.0153
7	0.0151	0.0178
8	0.0165	0.0157
9	0.0144	0.0170
10	0.0152	0.0153





```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

```
x = np.linspace(0, 100, 1000)
y = np.sin(x)
seq_length = 20
X, Y = [], []
for i in range(len(y)-seq_length):
    X.append(y[i:i+seq_length])
    Y.append(y[i+seq_length])
```

```
X = np.array(X).reshape(len(X), seq_length, 1)
Y = np.array(Y)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = Y[:split], Y[split:]
```

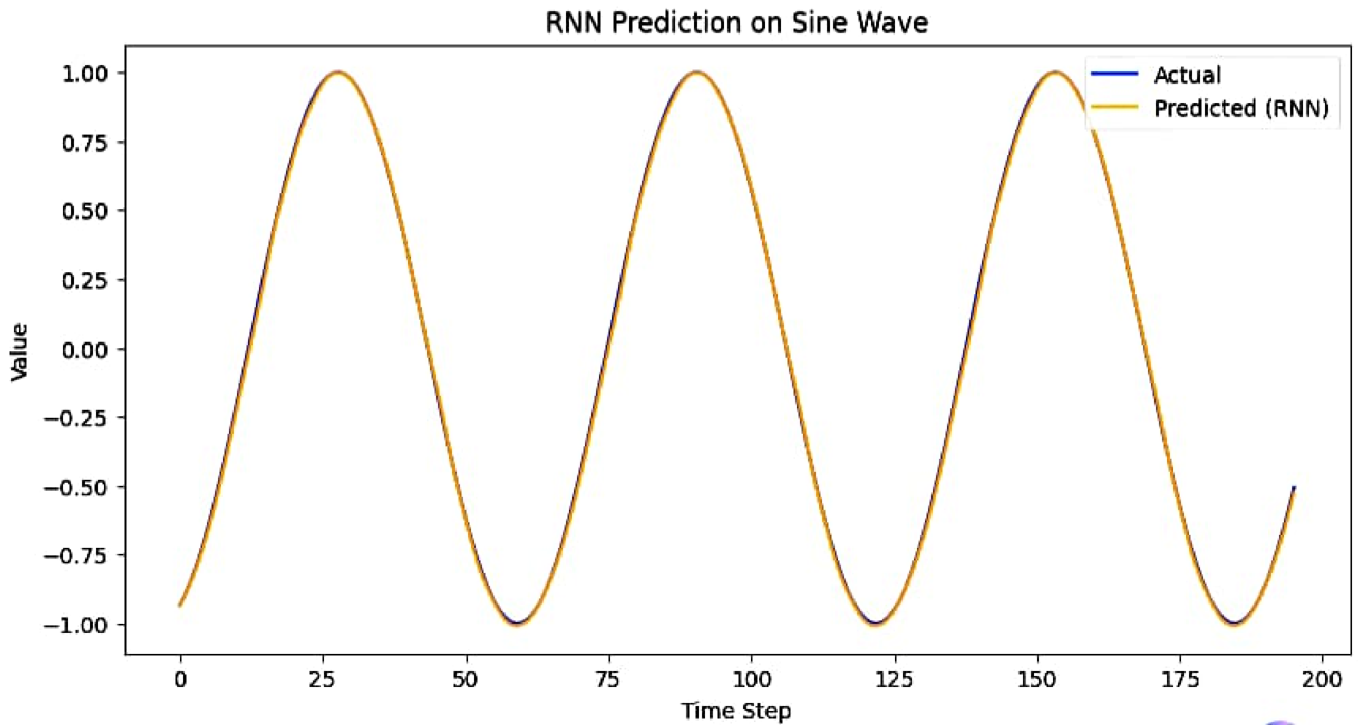
```
model = Sequential([
    SimpleRNN(100, activation='tanh', input_shape=(seq_length, 1), return_sequences=True),
    Dropout(0.2),
    SimpleRNN(50, activation='tanh'),
    Dense(1)
])
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
```

```
history = model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=32,
    validation_split=0.1,
    verbose=1
)
```

Commands + Code + Text ▷ Run all ▾

```
[ ]
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"MSE: {mse:.5f}, RMSE: {rmse:.5f}")
plt.figure(figsize=(10,5))
plt.plot(y_test, label='Actual', color='blue')
plt.plot(y_pred, label='Predicted (RNN)', color='orange')
plt.title('RNN Prediction on Sine Wave')
plt.xlabel('Time Step')
plt.ylabel('Value')
plt.legend()
plt.show()
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Training Loss', color='red')
plt.plot(history.history['val_loss'], label='Validation Loss', color='green')
plt.title('RNN Training Loss')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
MSE: 0.00013, RMSE: 0.01155
```


RNN Prediction on Sine Wave
Actual
Predicted (RNN)
Value
Time Step