REPORT ON TIME-SERIES DATA

1. How to apply RNNs to time-series data?

Data preparation: Make sure the data is arranged in time-step sequences. For example, make sliding windows of fixed-length sequences (e.g., 30 days) for forecasting future sales based on historical data.To increase the stability and effectiveness of model training, normalize or standardize your time-series data.

Model Definition: Make use of frameworks such as PyTorch or TensorFlow/Keras. Depending on your task, select an RNN layer type,
For brief sequences or simple dependencies, use SimpleRNN. Long Short-Term Memory (LSTM) is used to handle vanishing gradient problems and record long-term dependencies. An alternative to LSTM with a somewhat more straightforward design is the GRU (Gated Recurrent Unit). Set up the RNN input shape [batch_size, time_steps, features], where features stand for variables at each time step and time_steps is the sequence length.

Training: Assemble the model using the proper optimizer (like Adam) and loss function (like Mean Squared Error for regression).
Divide the data into test, validation, and training sets.
Use validation loss to track the model's performance as you train it over several epochs.

2. How to improve performance of the network, especially when dealing with time-series data?

Consider these techniques to improve an RNN's performance on time-series data:

Optimize Model Architecture: By layering layers, you can experiment with the RNN's depth and perhaps capture more intricate patterns.
To better handle long-term dependencies, try using LSTM or GRU layers rather than SimpleRNN.

Regularization: To avoid overfitting, especially when data is scarce, include dropout layers. Avoid high weight values by using L2 regularization (weight decay), which can aid with generalization.

Incorporate Attention Mechanisms: In order to enhance predictions in lengthy time-series data, an attention mechanism enables the network to concentrate on pertinent segments of the sequence.

Optimize training: To prevent overfitting, use adaptive optimizers such as Adam or RMSprop and consider learning rate schedules or early stopping.

3. How to apply different deep-learning layers to time-series data?

You can enhance model capabilities in time-series modeling by combining different deep-learning layers:

Convolutional Layers: Prior to input the time-series data into an RNN, utilize 1D convolutional layers to extract spatial (temporal) characteristics. This is frequently utilized in hybrid CNN-RNN models and is capable of efficiently capturing local patterns.

Dense layers: Dense layers, also known as fully connected layers, are added after RNN layers to process the features that have been retrieved and produce final predictions. With activation functions like linear activation for regression or softmax for classification, dense layers are frequently employed as output layers.

Dropout Layers: To lessen overfitting, add dropout layers after RNN or dense layers. This is particularly helpful when data is few.

Attention Layers: By assisting the model in selectively focusing on pertinent portions of the input over lengthy sequences, attention layers enhance performance on challenging time-series tasks.

Batch Normalization: By standardizing the input for every layer, batch normalization can stabilize and speed up training. In deeper networks, this is frequently utilized.

Model	Hidden Units	Training MAE	Validation MAE	Test MAE
Recurrent layer	32	7.0961	6.6685	6.65
Recurrent Layer	64	0.2539	0.2377	0.25
LSTM	16	0.2453	0.2678	0.25
1D CNN + LSTM		0.2565	0.2630	0.25

Conclusion:

As seen by the recurrent layer with 64 hidden units, which achieves a substantially lower MAE (around 0.25), in comparison to the recurrent layer with 32 hidden units, the results show that increasing the hidden units significantly improves model performance. With Training, Validation, and Test MAE values of approximately 0.25, both the LSTM with 16 hidden units and the recurrent layer with 64 hidden units exhibit optimal performance, indicating that they successfully capture the underlying patterns in the data. The lack of extra accuracy improvements offered by the 1D CNN + LSTM model suggests that recurrent layer or LSTM architectures, which are simpler, can accomplish comparable outcomes at a potentially lower computational cost. Thus, the best option would be either the recurrent layer with 64 hidden units or the LSTM with 16 hidden units.