RNN Assignment: Student Instructions v2.0

Recurrent Neural Networks vs Transformers for Time Series Prediction

Updated: Library-based Transformers Accepted

AIMLCZG511 Deep Neural Networks Course
BITS Pilani Work Integrated Learning Programme

Semester 1, 2025-26

CRITICAL - READ CAREFULLY

Submission Deadline: 08-Feb-26
Total Marks: 20 (Scaled to 10)
Submission Attempts: ONE only
Submission Mode: LMS only (NO email submissions)
File Format: .ipynb ONLY (NO zip files, NO data files)

1 Assignment Overview

1.1 Learning Objectives

By completing this assignment, you will:

- Build LSTM/GRU models for time series forecasting
- Use transformer architecture with positional encoding
- Understand attention mechanisms for sequential data
- Compare RNN vs Transformer approaches
- Apply time series preprocessing and evaluation techniques

1.2 What You Will Build

1. **RNN Model:** LSTM or GRU using Keras/PyTorch
2. **Transformer Model:** Encoder with attention using PyTorch/Keras libraries
3. **Comparison:** Analyze performance and architectural differences

1.3 Time Estimate

Expected Time: 8-10 hours total

- Dataset preparation and exploration: 2-3 hours
- LSTM/GRU implementation: 2-3 hours
- Transformer implementation: 3-4 hours
- Analysis and documentation: 1 hour

2 Submission Requirements

CRITICAL - READ CAREFULLY

AUTOMATIC ZERO MARKS if ANY of these are violated:

1. Wrong filename format
2. BITS ID mismatch (filename vs notebook)
3. Student name mismatch (LMS Name vs notebook)
4. Notebook not executed (no outputs visible)
5. Execution errors present
6. Positional encoding NOT implemented
7. Multi-head attention NOT implemented
8. Using pre-trained transformers (HuggingFace, TimeGPT, etc.)
9. Shuffling time series data (temporal order violated)
10. Missing JSON output

2.1 Filename Format

Required format: `<BITS_ID>_rnn_assignment.ipynb`

Examples:

- ✓ `2025AA05036_rnn_assignment.ipynb`
- ✗ `RNN_Assignment.ipynb`
- ✗ `2025AA05036_Assignment3.ipynb`
- ✗ `TimeSeriesProject.ipynb`

2.2 Student Information

Fill in the first cell of your notebook:

```
BITS ID: 2025AA01234
Name: JOHN DOE
Email: john.doe@wilp.bits-pilani.ac.in
Date: 15-01-2026
```

CRITICAL: BITS ID in filename MUST match BITS ID in notebook.

2.3 Execution Requirements

Before submission:

1. Click **Kernel → Restart & Run All**
2. Wait for ALL cells to execute
3. Verify ALL outputs are visible

4. Check for any error messages
5. Ensure JSON output is printed at the end

3 Technical Requirements

3.1 Framework Choice

Choose ONE framework (do NOT mix):

- **Option 1:** Keras/TensorFlow
- **Option 2:** PyTorch

3.2 Dataset Requirements

3.2.1 Allowed Datasets

Choose ONE from:

1. **Stock Prices** (daily/hourly closing prices)
2. **Weather Data** (temperature, humidity, pressure)
3. **Energy Consumption** (electricity/power usage)
4. **Sensor Data** (IoT sensor readings)
5. **Custom time series** (requires instructor approval)

3.2.2 Dataset Specifications

- **Minimum:** 1000 time steps
- **Train/Test Split:** 90/10 OR 85/15 (temporal split only)
- **Sequence Length:** 10-50 time steps (lookback window)
- **Prediction Horizon:** 1-10 time steps ahead
- **Features:** Univariate or multivariate

CRITICAL - READ CAREFULLY

TIME SERIES SPLIT - CRITICAL REQUIREMENT:
CORRECT (Temporal Split):

```
split_idx = int(len(data) * 0.9)
train_data = data[:split_idx]   # Earlier data
test_data = data[split_idx:]    # Later data
```

WRONG (Shuffled Split):

```
# DO NOT DO THIS - violates temporal order
train_test_split(X, y, shuffle=True)  # WRONG!
```

Why? Time series has temporal dependencies. Shuffling destroys the sequential structure.

4 Assignment Components

4.1 Part 1: LSTM/GRU Model (5 marks)

4.1.1 Requirements

Build LSTM or GRU (choose ONE) using Keras/PyTorch with:

- At least 2 stacked recurrent layers
- Output layer for prediction
- Standard training methods (model.fit() or PyTorch loop)
- Track initial and final loss

4.1.2 Grading Breakdown

- Architecture with stacked layers ($\geq$2): 2 marks
- Model properly compiled/configured: 1 mark
- Training completed with loss tracking: 1 mark
- All 4 metrics calculated correctly: 1 mark

4.2 Part 2: Transformer Model (5 marks)

CRITICAL - READ CAREFULLY

IMPLEMENTATION OPTIONS (Choose ONE):

1. **PyTorch Library:** Use `torch.nn.TransformerEncoder`
2. **Keras Library:** Use `keras.layers.MultiHeadAttention`
3. **Manual:** Implement Q, K, V, scaled dot-product yourself

MANDATORY FOR ALL OPTIONS:

- Positional encoding MUST be added (custom or built-in)

PROHIBITED:

- Pre-trained transformers (HuggingFace, TimeGPT, Chronos, etc.)
- Skipping positional encoding

4.2.1 Grading Breakdown

- Positional encoding added: 1 mark
- Multi-head attention (library OR manual): 2 marks
- Training completed with loss tracking: 1 mark
- All 4 metrics calculated correctly: 1 mark

4.2.2 Positional Encoding

Required Implementation (for all options):

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

where pos is position, i is dimension index, and d_{model} is model dimension.

4.2.3 Multi-Head Attention

Implementation Options:

Option A - PyTorch Library:

- Use `torch.nn.TransformerEncoder` with custom positional encoding
- Specify `nhead` parameter (must be > 1)
- Add positional encoding before transformer layers

Option B - Keras Library:

- Use `keras.layers.MultiHeadAttention`
- Specify `num_heads` parameter (must be > 1)
- Add custom positional encoding to input

Option C - Manual Implementation:

1. **Q, K, V Projections:** Linear transformations
2. **Scaled Dot-Product:**

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

3. **Multi-Head:** Multiple attention heads concatenated
4. **Output Projection:** Final linear layer

4.3 Part 3: Training Process (4 marks)

4.3.1 Convergence Requirements

Loss reduction is calculated as:

$$\text{Reduction } \% = \frac{\text{Initial Loss} - \text{Final Loss}}{\text{Initial Loss}} \times 100$$

Grading:

- $\geq$50% reduction: 2 marks
- $\geq$20% reduction: 1 mark
- $<$20% reduction: 0 marks

This applies to BOTH models (2 marks each = 4 marks total).

4.4 Part 4: Metrics Calculation (2 marks)

4.4.1 Required Metrics (ALL 4 for BOTH models)

1. **MAE** (Mean Absolute Error): Must be > 0
2. **RMSE** (Root Mean Squared Error): Must be > 0
3. **MAPE** (Mean Absolute Percentage Error): Must be > 0
4. **R^2 Score** (Coefficient of Determination): Must be in $[-1, 1]$

Formulas:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

$$\text{R}^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

4.4.2 Primary Metric Selection

Choose ONE as your primary metric and justify:

- **MAE:** For average error magnitude
- **RMSE:** For penalizing large errors
- **MAPE:** For percentage error (scale-independent)

4.5 Part 5: Analysis (2 marks)

4.5.1 Requirements

Write an analysis (maximum 200 words guideline) addressing:

1. Which model performed better and by how much?
2. RNN vs Transformer architecture advantages
3. Impact of attention mechanism vs recurrent connections
4. Long-term dependency handling comparison
5. Computational cost (training time, parameters)
6. Convergence behavior differences

4.5.2 Grading Criteria

- Covers 5+ key topics with depth: 2 marks
- Covers 3-4 key topics: 1 mark
- Covers <3 topics or superficial: 0 marks

Note: Word count >200 will generate a warning but NO marks deduction.

4.6 Part 6: Code Structure (2 marks)

4.6.1 Requirements

- Both RNN and Transformer properly implemented: 1 mark
- JSON output with correct structure: 1 mark

5 JSON Output Format

Important Information

The autograder relies on exact JSON field names. Use the provided template's `get_assignment_results()` function WITHOUT modification.

6 Common Mistakes to Avoid

Important Information

Top reasons for losing marks:

1. Not adding positional encoding to transformer (automatic 0 for Transformer)
2. Not using multi-head attention (automatic 0 for Transformer)
3. Using pre-trained transformers (HuggingFace, TimeGPT) (automatic 0 for Transformer)
4. Shuffling time series data (violates temporal order)
5. Only 1 RNN layer instead of $\geq$2 stacked layers
6. Missing metrics (only 3 out of 4)
7. Metrics outside valid ranges
8. Not tracking initial/final loss
9. JSON output missing or incorrect field names (very low marks)

6.1 Implementation Mistakes

6.1.1 Wrong: Single RNN Layer

```
model = Sequential([
    LSTM(64),  # WRONG - only 1 layer
    Dense(1)
])
```

6.1.2 Correct: Stacked RNN Layers

```
model = Sequential([
    LSTM(64, return_sequences=True),  # Layer 1
    LSTM(32),  # Layer 2 - CORRECT
    Dense(1)
])
```

6.1.3 Wrong: Using Pre-trained Transformer

```
# WRONG - using pre-trained from HuggingFace
from transformers import AutoModel
model = AutoModel.from_pretrained("timegpt")
```

6.1.4 Correct: Transformer with Library

```
# CORRECT - using PyTorch library with positional encoding
class TransformerModel(nn.Module):
    def __init__(self, ...):
        self.pos_encoder = PositionalEncoding(...)  # Add PE
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=64, nhead=4, ...)  # Library
        self.transformer = nn.TransformerEncoder(encoder_layer, ...)
```

7 Time Series Preprocessing

7.1 Normalization

Always normalize your data:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)
```

7.2 Sequence Creation

Create sliding windows:

```
def create_sequences(data, seq_length, horizon):
    X, y = [], []
    for i in range(len(data) - seq_length - horizon + 1):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length:i+seq_length+horizon])
    return np.array(X), np.array(y)
```

8 Helpful Tips

8.1 Debugging Tips

8.1.1 If Training is Too Slow

- Reduce sequence length
- Use fewer training samples for testing
- Reduce number of epochs during development
- Reduce model complexity (fewer layers/units)
- Use GPU (use Virtual Lab / Google Colab)

8.1.2 If Predictions are Poor

- Check data normalization
- Verify sequence creation (no data leakage)
- Try longer sequence length
- Increase model capacity (more units/layers)
- Train for more epochs
- Check for NaN values in data

8.1.3 If Attention is Not Working

- Verify Q, K, V dimensions match
- Check scaling: divide by $\sqrt{d_k}$
- Ensure softmax is applied to scores
- Verify multi-head concatenation
- Print attention weights to debug

Helpful Tips

Before submission checklist:

1. ❑ Filename is `<BITS_ID>_rnn_assignment.ipynb`
2. ❑ BITS ID in filename matches notebook
3. ❑ Student name matches the name in LMS
4. ❑ Ran Kernel → Restart & Run All
5. ❑ All outputs visible
6. ❑ No execution errors
7. ❑ LSTM/GRU has ≥2 stacked layers
8. ❑ Positional encoding implemented
9. ❑ Multi-head attention implemented
10. ❑ Temporal split used (NO shuffling)
11. ❑ All 4 metrics calculated for both models
12. ❑ JSON output printed at the end
13. ❑ Analysis written
14. ❑ Prediction plots created
15. ❑ Environment screenshot included

9 Evaluation Rubric Summary

Component	Marks	Key Requirements
LSTM/GRU	5	Stacked ($\geq$2), compile, train, metrics
Transformer	5	Pos. encoding, attention, train, metrics
Training	4	Convergence $\geq$20% for both
Metrics	2	All 4 metrics, valid ranges
Analysis	2	Quality-based, 5+ topics
Code Structure	2	Implementations + JSON
TOTAL	**20**	**Scaled to 10**

Marks Distribution

10 FAQs

1. Can I use both LSTM and GRU?
   No. Choose ONE (LSTM or GRU), not both.

2. Can I use HuggingFace Transformers library?
   No. HuggingFace models are pre-trained. Use PyTorch `nn.TransformerEncoder` or Keras `MultiHeadAttention` instead.

3. Can I use `torch.nn.TransformerEncoder`?
   Yes! Use library transformers (`nn.TransformerEncoder` or `MultiHeadAttention`), but you MUST add positional encoding yourself.

4. Can I shuffle my time series data?
   No. Time series must maintain temporal order. Use chronological split.

5. What if I have multivariate time series?
   That's fine. Your model should handle multiple features per time step.

6. Do I need layer normalization in Transformer?
   Recommended but not mandatory. Focus on attention and positional encoding.

7. Can I use single-head attention?
   You'll lose marks. Multi-head attention (>1 head) is required for full credit.

8. Should I inverse transform predictions?
   Yes, if you normalized your data. Report metrics on original scale.

9. Do I need to submit my dataset?
   No. Submit ONLY the notebook file. Load data from public sources or URLs.

10. Can I resubmit if I make a mistake?
    No. You have ONE submission attempt only. Test thoroughly before submitting.

11. Will there be partial credit?
    Yes. Each section is graded independently. Even if one part fails, you get credit for completed parts.

12. What if my notebook doesn't open?
    You will receive 0 marks. Test that your file opens correctly before submission.

11 Support and Resources

11.1 Getting Help

- **Course Forum:** Post questions on LMS discussion board
- **Documentation:** Keras/PyTorch official docs

11.2 Useful Resources

- Keras LSTM: `https://keras.io/api/layers/recurrent_layers/`
- PyTorch RNN: `https://pytorch.org/docs/stable/nn.html`
- Attention Paper: "Attention Is All You Need" (Vaswani et al., 2017)

Good Luck!

Focus on correct implementation of attention and positional encoding.
These are the key innovations you need to demonstrate understanding of.