RNNs with PyTorch

Prashant Kumar Singh

13 March 2025

Project Overview

This project explores Recurrent Neural Networks (RNNs) using PyTorch for time series forecasting and sequence modeling. The notebook demonstrates the implementation of multiple RNN architectures including:

- Fully Connected Network (FCN)
- Simple RNN
- Deep RNN
- LSTM
- GRU
- Multi-step RNN

The project includes both synthetic time series data and real-world stock price prediction using historical Apple Inc. (AAPL) data.

Objectives

- Understand RNN architectures and their components.
- Learn to implement various sequence models in PyTorch.
- Apply models to predict future time steps in a sequence.
- Explore multi-step forecasting for practical scenarios.

Setup

- Recommended to run the notebook on **Google Colab** with GPU enabled.
- Required libraries include `torch`, `numpy`, `matplotlib`, `pandas`, `yfinance`, and `sklearn`.

Dataset

Synthetic Time Series

Generated using a combination of sine waves and noise:

$$x(t) = 0.5\sin((t - o_1)(10f_1 + 10)) + 0.2\sin((t - o_2)(20f_2 + 20)) + 0.1\epsilon$$

Stock Prices (Apple Inc.)

Historical stock prices fetched via `yfinance` for dates Jan 1, 2010 to Jan 1, 2021. The `Close` price is used for prediction.

Models Implemented

Fully Connected Network

Treats each time step as independent:

```
class FCN(nn.Module):
    def __init__(self, n_steps):
        super().__init__()
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(n_steps, 1)
    def forward(self, x):
        x = self.flatten(x)
        return self.fc(x)
```

Simple RNN

```
class SimpleRNNModel(nn.Module):
    def __init__(self, input_size=1, hidden_size=1):
        super().__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=
           True)
        self.fc = nn.Linear(hidden_size, 1)
    def forward(self, x):
        out, _ = self.rnn(x)
        out = out[:, -1, :]
        return self.fc(out)
```

Deep RNN, LSTM, GRU, Multi-step RNN

Stacked RNN layers, LSTM, and GRU models were implemented for enhanced learning of sequential patterns. Multi-step forecasting predicts several future time points simultaneously.

Training and Evaluation

- Loss function: Mean Squared Error (MSE)
- Optimizer: Adam
- Models trained for varying epochs depending on complexity.
- Visualizations of predicted vs actual series for evaluation.

Key Findings

- RNN-based models outperform FCN on sequential data due to memory of past time steps.
- LSTM and GRU handle long-term dependencies better than simple RNNs.
- Multi-step prediction is crucial for real-world forecasting tasks like stock prices.

Usage

1. Run the notebook cells sequentially to train models and generate predictions.
2. Modify parameters such as `hidden_size`, `num_layers`, and `n_steps` for experimentation.
3. Use `plot_series` and `plot_multiple_forecasts` functions to visualize results.

References

- PyTorch Documentation: `https://pytorch.org/docs/stable/index.html`
- Yahoo Finance: `https://finance.yahoo.com/`