Recurrent Neural Network (RNN) for Sequence Prediction

Overview

This project implements a Recurrent Neural Network (RNN) using PyTorch to model sequential data. RNNs are designed to recognize patterns in sequences of data, making them ideal for tasks such as time series prediction, natural language processing, and speech recognition. In this implementation, the RNN is trained to predict a continuous output based on input sequences.

Why Use This Model

Recurrent Neural Networks are particularly effective for modeling time-dependent data due to their ability to maintain a hidden state that captures information about previous inputs. This characteristic makes RNNs suitable for applications where the context of previous data points is crucial for accurate predictions.

Prerequisites

Before running the code, ensure you have the following Python packages installed:

- `torch`
- `numpy`

You can install these packages using pip:

```
pip install torch numpy
```

Files Included

- `rnn_model.py` : Contains the implementation of the RNN model, including data preparation, model definition, training loop, and evaluation.

Code Description

1. **Data Preparation**: The dataset consists of random sequences generated using NumPy. Each sequence has a length of 10, with 5 features per time step. The target output is a continuous value for each time step.

   ```
    import torch
   import torch.nn as nn
   import torch.optim as optim
   import numpy as np

   # Hyperparameters
   seq_length = 10
   batch_size = 16
   input_size = 5
   output_size = 1
   ```

```
# Generate random input and output sequences
X = torch.randn((batch_size, seq_length, input_size))
y = torch.randn((batch_size, seq_length, output_size))
```

2. **Model Definition**: The `BasicRNN` class defines the RNN architecture, which includes an RNN layer followed by a fully connected layer.

```
class BasicRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(BasicRNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc(out)
        return out
```

3. **Model Training**: The model is trained using the Adam optimizer and Mean Squared Error loss function. Training runs for 50 epochs, with loss printed every 10 epochs.

```
 # Initialize model, loss function, and optimizer
model = BasicRNN(input_size, hidden_size=8, output_size=output_size)
optimizer = optim.Adam(model.parameters(), lr=0.01)
loss_fn = nn.MSELoss()

# Training loop
for epoch in range(50):
    optimizer.zero_grad()
    output = model(X)
    loss = loss_fn(output, y)
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch+1}/50], Loss: {loss.item():.4f}")
```

Expected Outputs

The model outputs a sequence of predicted values for each input sequence. During training, the loss is printed every 10 epochs to monitor the model's performance.

Use Cases

- **Time Series Prediction**: Predicting future values based on historical data.
- **Natural Language Processing**: Modeling sequences of words or characters for tasks like language modeling or text generation.
- **Speech Recognition**: Processing sequences of audio features to recognize spoken words.

Advantages

- **Sequential Data Handling**: RNNs are specifically designed to handle sequential data, making them suitable for time-dependent tasks.

- **Parameter Sharing**: RNNs share parameters across time steps, reducing the number of parameters and computational complexity.

Future Enhancements

- **Bidirectional RNNs (BRNNs)**: Implementing BRNNs to capture context from both past and future inputs.
- **Long Short-Term Memory (LSTM) Networks**: Replacing the basic RNN with LSTMs to mitigate issues like vanishing gradients and improve long-term dependency modeling.
- **Attention Mechanisms**: Incorporating attention mechanisms to allow the model to focus on specific parts of the input sequence.

References

- PyTorch RNN Documentation
- Understanding LSTM Networks
- Attention Is All You Need