Sequence to Sequence Recurrent Neural Network with BackPropagation Through Time (BPTT)

This script implements a simple recurrent neural network (RNN) with a single hidden layer trained using a simplified recurrent architecture that processes sequences step-by-step with immediate weight updates. The RNN accepts a dynamic excitation time history and returns the response of a SDOF oscillator to this excitation. For this purpose it is trained using an amplification-enhanced white noise excitation, which includes a wide range of frequency content which is seen by the RNN during training. After training, the prediction of the RNN to a linear combination of sinusoidal time histories is tested. Successful agreement exists between the dynamic response of the SDOF oscillator predicted by the RNN and calculated rigorously using the Newmark time integration algorithm. A similar job for seismic dynamic response prediction of a slope system is presented by Huang et al. (2021) (see README.md).

Contents

Set random seed for reproducibility

```
rng(0);
```

SDOF system parameters

Set the properties of the SDOF system considered. These properties are identical for training, validation and testing the RNN Mass

```
m = 1.0;
% Stiffness
k = 100.0;
% Damping
c = 2.0;
% Time step
dt = 0.02;
% Natural frequency
omega_n = sqrt(k/m);
f_n = omega_n / (2*pi);
% Damping ratio
zeta = c / (2*sqrt(m*k));
```

RNN Parameters

Set the various hyperparameters of the RNN

```
sequence_length = 100;
batch_size = 32;
window_length = 20;
hidden_units = 128;
```

```
learning_rate = 0.005;
val_size = 0.2;
epochs = 300;
patience = 25;
```

Prepare training data for RNN

Generate training data (white noise excitation)

```
[input_train, output_train] = example_generate_training_data(10000, dt, m, k, c);
% Prepare training data
[X_train, X_val, y_train, y_val, input_scaler, output_scaler] = data_prepare_training(...
    input_train, output_train, sequence_length, val_size);
% Initialize RNN weights
[W1, W2, W3, b1, b2] = rnn_initialize_weights(1, hidden_units, 1);
```

RNN Training

Train the RNN model

```
[W1, W2, W3, b1, b2, train_loss_history, val_loss_history] = rnn_train(...
    X_train, y_train, X_val, y_val, W1, W2, W3, b1, b2, ...
    learning_rate, epochs, batch_size, window_length, patience);
```

Testing RNN on validation data

Evaluate model on validation data

```
[mse, mae, predictions, true_values] = rnn_evaluate(...
    X_val, y_val, W1, W2, W3, b1, b2, output_scaler);
```

Testing RNN on testing data

Generate testing data (sinusoidal excitation at various frequencies)

```
[input_test, output_test] = example_generate_testing_data(10000, dt, m, k, c);
% Prepare the testing data using the same scalers from training
[X_test, y_test] = data_prepare_testing(...
    input_test, output_test, sequence_length, input_scaler, output_scaler);
% Evaluate on testing data
[mse_test, mae_test, predictions_test, true_values_test] = rnn_evaluate(...
    X_test, y_test, W1, W2, W3, b1, b2, output_scaler);
```

Plotting

Plot comprehensive results comparing both datasets

```
plot_comparison(train_loss_history, val_loss_history, ...
    true_values, predictions, true_values_test, predictions_test, m, k, c);
```

Published with MATLAB® R2024b