Gravitational Wave Event Detection: Developing Convolutional Neural Networks and Recurrent Neural Networks for Gravitational Wave Data Analysis

Shufan Dong

1Class of 2026, Bronx High School of Science, NY, USA.
dongs1@bxscience.edu

Abstract

Gravitational wave (GW) data analysis has evolved significantly with advancements in machine learning (ML) techniques, particularly convolutional neural networks (CNNs) and Recurrent neural networks (RNNs). This paper presents a comprehensive approach to developing both CNN and RNN models for the analysis of GW data. We convert the time-series strain data into spectrograms for the 2D CNN while retaining the time-series format for the 1D CNN and RNN models (LSTM and GRU), allowing the input of both data representations into the ML models. The overall procedure for training the 1D CNN and RNN models includes data segmentation, time-series data reshaping, data augmentation, model training, and model evaluation and visualization. Then, a similar process is applied to training the 2D CNN model, which includes data segmentation, spectrogram data generation and reshaping, data augmentation, model training, and model evaluation and visualization. Our results demonstrate the use of these methods in accurately classifying GW events, highlighting the potential for further applications in astrophysical research.

Contents

1 Introduction .......................................................... 2

2 Importing Libraries .................................................. 2

3 Data Segmentation and Labeling ................................. 3

4 Data Preparation ...................................................... 5

4.1 Time-series Data Reshaping for 1D CNN and RNNs .......... 5

4.2 Spectrogram Generation for 2D CNN ....................... 5
1 Introduction

The detection and analysis of gravitational waves (GWs) have opened a new window into the universe, allowing us to observe astrophysical phenomena that were previously inaccessible. The Laser Interferometer Gravitational-Wave Observatory (LIGO) and its international partners have made continuous efforts in this field, detecting numerous GW events from merging black holes and colliding neutron stars. As the volume of GW data increases, there is a growing need for efficient and accurate analysis techniques. This paper explores the use of CNNs and RNNs for classification purposes, focusing on 1D CNN, 2D CNN, LSTM, and GRU models. The preprocessing of GW data was done previously already, and you can refer to [49] for more details.

2 Importing Libraries

Necessary libraries are required for the CNN and RNN models in this paper, utilizing the TensorFlow tool for modeling our CNNs and RNNs.
Figure 1: The packages imported are added in addition to the imported libraries discussed in [49], which contains the basic libraries required to preprocess and manipulate GW data.

3 Data Segmentation and Labeling

The continuous GW strain data is split into smaller, manageable segments and labeled appropriately. This step is critical for preparing the dataset for supervised learning, allowing the model to learn based on discoverable patterns.
Figure 2: The function `create_segments_and_labels` is used to split the strain data into segments of 2 seconds each, starting at `t_start` (start of GW150914 event) and sampled at `fs` Hz (4096 Hz).

```python
def create_segments_and_labels(strain, event_time, window_size, sample_rate):
    # Resample strain to desired sample rate (if necessary)
    strain = strain.resample(sample_rate)

    # Def segments and labels ls
    segments = []
    labels = []

    # Calc # of samples per segment
    segment_length = int(window_size * sample_rate)

    # Create segments and labels
    for i in range(0, len(strain) - segment_length, segment_length):
        segment = strain[i:i + segment_length]
        segments.append(segment)

        # label based on event presence
        if segment.times.value[0] <= event_time <= segment.times.value[-1]:
            labels.append(1)  # Event present
        else:
            labels.append(0)  # No event

    # Convert to np arrays
    segments = np.array(segments)
    labels = np.array(labels)

    return segments, labels

segments, labels = create_segments_and_labels(strain, t_start, 2, fs)
```

Figure 3: The shape of GW data’s segments and labels.
4 Data Preparation

4.1 Time-series Data Reshaping for 1D CNN and RNNs

To ensure the compatibility of the time-series data for 1D CNN and RNNs, time-series data is reshaped to include an extra dimension.

```python
# Reshape segments for 1D CNN
segments = segments.reshape((segments.shape[0], segments.shape[1], 1))

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(segments, labels, test_size=0.2, random_state=42)
```

Figure 4: The segment data is reshaped with an additional dimension of 1. Then, the data is split into the training set (80% of the data) and the testing set (20% of the data).

```
Reshaped segments shape: (2047, 8192, 1)
```

Figure 5: The shape of the input data for 1D CNN and RNNs.

4.2 Spectrogram Generation for 2D CNN

To examine the spatial feature extraction capabilities of 2D CNN, time-series data is converted into spectrograms, which provide a frequency domain representation of the data.

```python
# Generate spectrograms for each segment
def generate_spectrogram(segment, sample_rate):
    f, t, Sxx = spectrogram(segment, sample_rate)
    return Sxx

spectrograms = np.array([generate_spectrogram(segment, fs) for segment in segments])

# Reshape spectrograms for 2D CNN
spectrograms = spectrograms[..., np.newaxis]  # Add a channel dim

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(spectrograms, labels, test_size=0.2, random_state=42)
```

Figure 6: The `generate_spectrogram` function converts each time-series segment into a spectrogram, and the spectrograms are then reshaped to include a channel dimension for compatibility with 2D CNN input. Then, the data is split into the training set (80% of the data) and testing set (20% of the data).
5 Data Augmentation

To prevent overfitting and improve generalization, data augmentation techniques are applied to the training data.

```python
def augment_data(data, labels):
    augmented_data = []
    augmented_labels = []
    for d, l in zip(data, labels):
        augmented_data.append(d)
        augmented_labels.append(l)
        augmented_data.append(np.flip(d, axis=0))
        augmented_labels.append(l)
        noise = np.random.normal(0, 0.1, d.shape)
        augmented_data.append(d + noise)
        augmented_labels.append(l)
    return np.array(augmented_data), np.array(augmented_labels)
```

Figure 8: The `augment_data` function artificially increases the size of the training dataset by introducing variability.

Figure 9: The shape of the data for 1D CNN and RNNs before and after augmentation.

Figure 10: The shape of the data for 2D CNN before and after augmentation.
6 Model Training

6.1 1D CNN
A 1D CNN model is constructed and trained on the augmented time-series data.

![Table of layers and parameters]

Figure 11: The 1D CNN model processes the time-series data directly, using convolutional layers to extract temporal features, pooling layers to reduce dimensionality, dense layers to classify event presence, and a dropout layer to prevent overfitting.

6.2 2D CNN
A 2D CNN model is built and trained on the augmented spectrogram data.
Figure 12: The 2D CNN model consists of convolutional layers for feature extraction, pooling layers for dimensionality reduction, and dense layers for event classification. A dropout layer is added to help prevent overfitting.

### 6.3 LSTM

An LSTM model is constructed and trained on the augmented time-series data.
Figure 13: The LSTM model processes the time-series data directly, using two LSTM layers for feature extraction, two dropout layers to prevent overfitting, and a dense layer to classify event presence. For quicker model training, the size of the data for LSTM is resampled to four times less than the data for 1D and 2D CNN.

### 6.4 GRU

A GRU model is constructed and trained on the augmented time-series data.
Figure 14: The GRU model processes the time-series data directly, using two GRU layers for feature extraction, two dropout layers to prevent overfitting, and a dense layer to classify event presence. For quicker model training, the size of the data for LSTM is resampled to four times less than the data for 1D and 2D CNN.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>gru (GRU)</td>
<td>(None, 2048, 64)</td>
<td>12864</td>
</tr>
<tr>
<td>dropout_2 (Dropout)</td>
<td>(None, 2048, 64)</td>
<td>0</td>
</tr>
<tr>
<td>gru_1 (GRU)</td>
<td>(None, 64)</td>
<td>24960</td>
</tr>
<tr>
<td>dropout_3 (Dropout)</td>
<td>(None, 64)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 1)</td>
<td>65</td>
</tr>
</tbody>
</table>

7 Visualization and Evaluation

To examine the CNNs and RNNs more closely, the training and validation loss and accuracy are plotted to visualize the training process over epochs and determine the performance of each model.
7.1 1D CNN

Figure 15: These plots show the training history of the 1D CNN, including the test loss and accuracy evaluation.

7.2 2D CNN

Figure 16: These plots show the training history of the 2D CNN, including the test loss and accuracy evaluation.
7.3 LSTM

Figure 17: These plots show the training history of the LSTM, including the test loss and accuracy evaluation.

7.4 GRU

Figure 18: These plots show the training history of the GRU, including the test loss and accuracy evaluation.
8 Results and Discussion

Both CNN and RNN models showed promising results in classifying GW data based on event presence. The 1D CNN and RNNs effectively captured temporal patterns directly from the reshaped time-series data, while the 2D CNN used the frequency domain data representation provided by spectrograms. The models were evaluated based on test accuracy and loss, demonstrating their application in real-time GW event detection.

9 Conclusion and Outlook

This study highlights the effectiveness of CNNs and RNNs in analyzing GW data, with both 1D and 2D CNNs, LSTM, and GRU producing satisfactory results. Future work will involve exploring more advanced ML models applicable to the means of detecting GW event presence. The integration of these ML techniques into the GW analysis will enhance our ability to detect and interpret these cosmic events, contributing to our understanding of the universe and the future outlook of astrophysical research.

References


