Assignment 4: Neural Networks Regression for Gravitational Wave Signal Prediction

Jess Ross u7278803

October 24, 2024

1 Introduction

This project addresses the problem of predicting gravitational wave signals using a neural network. The goal is to train a regression model that learns to map the time series data (input) to the corresponding signal amplitude (output). The network is evaluated based on its ability to closely match the true signal.

1.1 Model Description

The model uses a simple feedforward neural network with multiple hidden layers to approximate the function that describes the underlying relationship between time and signal amplitude. This is treated as a time-series regression task where the network learns patterns from noisy input data to predict smooth gravitational wave signals.

1.2 Key Objectives

- Minimize the loss function, which in this case is Mean Squared Error (MSE), to bring the predicted signal as close as possible to the true signal.
- Explore the impact of different learning rates ($\eta = 0.1$) and ($\eta > 0.1$) on the training process. Specifically, we used learning rates of 0.1 and 0.5.

2 Graphing the Network Structure

The feedforward neural network has the following structure:

- Input layer: 1 neuron for time
- Two hidden layers with ReLU activation (5 neurons each)
- Output layer: 1 neuron for signal amplitude prediction

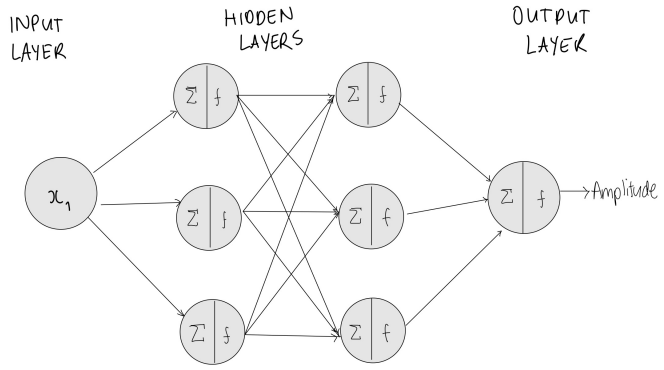

Figure 1: The structure of the neural network used for the regression task.

3 Equation of Gradients for All Neurons

The gradients of the neurons are computed during backpropagation. For each weight (w_{ij}) between neuron (i) and neuron (j), the gradient is calculated as follows:

3.1 Equations of Gradients for All Neurons

Consider a feedforward neural network where:

- (X) is the input, represented as an array.
- (h_1) and (h_2) are the neurons in the first and second hidden layers respectively.
- (O) is the output neuron.
- The weights between layers are denoted as (W_1, W_2, W_3).
- (b_1, b_2, b_3) are the biases for each layer.

3.1.1 Forward Propagation

1. Input to First Hidden Layer
   - $z_1 = W_1 X + b_1$
   - $h_1 = \text{ReLU}(z_1)$
2. First Hidden Layer to Second Hidden Layer
   - $z_2 = W_2 h_1 + b_2$

- $h_2 = \text{ReLU}(z_2)$

3. Second Hidden Layer to Output
   - $z_3 = W_3 h_2 + b_3$
   - $O = z_3$ (no activation at output)

3.1.2 Gradient Calculations

During backpropagation, the goal is to compute the gradient of the loss with respect to each weight, which is used to update the weights during training.

Gradient at the Output Layer: The loss (L) is defined as the Mean Squared Error (MSE) between the predicted output (O) and the true output (y):

$$L = \frac{1}{n} \sum_{i=1}^{n} (O_i - y_i)^2$$

The gradient of the loss with respect to O:

$$\frac{\partial L}{\partial O} = 2(O - y)$$

Since $O = z_3$, the gradient with respect to z_3:

$$\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial O} \cdot 1 = 2(O - y)$$

Gradient at Second Hidden Layer:

$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial z_3} \cdot \frac{\partial z_3}{\partial W_3} = 2(O - y) \cdot h_2$$

Similarly, for the bias:

$$\frac{\partial L}{\partial b_3} = 2(O - y)$$

Next, we calculate the gradient with respect to the output of the second hidden layer (h_2):

$$\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial z_3} \cdot W_3 = 2(O - y) \cdot W_3$$

Gradient at First Hidden Layer: For the first hidden layer, the gradient is similar, but we account for the ReLU activation function:

The gradient with respect to the input to the second hidden layer (z_2):

$$\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial h_2} \cdot \text{ReLU}'(z_2)$$

Where:

$$\text{ReLU}'(z_2) = \begin{cases} 1, & \text{if } z_2 > 0 \\ 0, & \text{otherwise} \end{cases}$$

Then the gradient with respect to the weight (W_2):

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial z_2} \cdot h_1$$

The gradient with respect to the bias (b_2):

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2}$$

Lastly, for the output of the first hidden layer (h_1):

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial z_2} \cdot W_2$$

Gradient at Input Layer: For the input layer, we apply the chain rule similarly:

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial h_1} \cdot \text{ReLU}'(z_1)$$

Then the gradient with respect to the weight (W_1):

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial z_1} \cdot X$$

And for the bias (b_1):

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1}$$

Weight Updates: Using gradient descent, the weights are updated as:

$$W = W - \eta \cdot \frac{\partial L}{\partial W}$$

where (η) is the learning rate.

4 Results

4.1 Predicted Signal vs True Signal

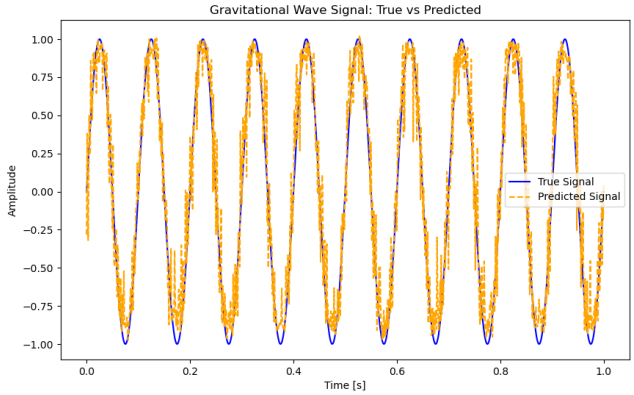

Figure 2: The predicted signal using the neural network against a test signal of constant amplitude with simulated noise.

5 Tables of Weights, Gradients, Outputs, and Loss

The tables below display the weights, gradients, outputs, and loss values recorded over 10 steps for two different learning rates (0.1 and 0.5):

Table 1: Dataset 1

Step	Weights (Layer 1)	Gradients (Layer 1)	Output
1	[0.46205887, -0.30744773, -0.34176743, -0.47089744, 0.5945953]	[0.25715047, 0.46112737, 0.03526331, -0.2049351, -0.07181387]	0.344097
2	[0.46111813, -0.36923525, -0.31679583, -0.52524024, 0.59524906]	[0.00188149, 0.12357503, -0.0499432, 0.1086856, -0.00130749]	0.227914
3	[0.40387967, -0.51096404, -0.32728934, -0.53671205, 0.67177147]	[0.11447691, 0.28345764, 0.02098702, 0.02294361, -0.15304479]	0.651483
4	[0.41320205, -0.6342356, -0.35122865, -0.5618635, 0.68156654]	[-0.01864476, 0.24654323, 0.04787862, 0.05030286, -0.01959016]	0.621475
5	[0.4198923, -0.5832042, -0.3460711, -0.5215145, 0.68272126]	[-0.01338052, -0.10206284, -0.01031511, -0.0806979, 0.00673153]	1.501322
6	[0.42400452, -0.7261559, -0.3199906, -0.6035096, 0.70655936]	[-0.00822445, 0.28590336, -0.052161, 0.16399033, -0.02656331]	0.628469
7	[0.38420963, -0.47328028, -0.23714055, -0.5076117, 0.7051711]	[0.07958978, -0.5057512, -0.16570011, -0.19179572, 0.03461538]	2.087253
8	[0.37555736, -0.4588536, -0.20111224, -0.43570772, 0.71963525]	[0.01730452, -0.02885338, -0.07205664, -0.00891549, 0.02677403]	1.691392
9	[0.38488817, -0.44256783, -0.2280885, -0.46237984, 0.7225137]	[-0.02172423, 0, 0, 0, 0]	0.035883
10	[0.40648162, -0.4588536, -0.20111224, -0.43570772, 0.71963525]	[-0.04012428, 0, 0, 0, 0]	0.083108

Table 2: Dataset 2

Step	Weights (Layer 1)	Gradients (Layer 1)	Output	Lo
1	[0.8669659, -0.01021131, 0.8163032, 0.01710683, 0.22552483]	[-0.01474498, 0.08346401, -0.14783329, -0.02716123, 0.04551014]	0.579949	0.3
2	[0.8692773, -0.03989618, 0.82959145, 0.01932875, 0.22210893]	[-0.02311431, 0.29684862, -0.13288252, -0.02221917, 0.03415895]	0.723741	0.2
3	[0.8725587, -0.04872571, 0.84388673, 0.02153953, 0.21897666]	[-0.03281416, 0.08829531, -0.14295293, -0.02210785, 0.03132265]	0.842752	0.2
4	[0.8756423, -0.06438935, 0.8548361, 0.02304757, 0.2169649]	[-0.03083608, 0.15663643, -0.10949396, -0.01508038, 0.02011762]	1.003722	0.1
5	[0.8789087, -0.07180712, 0.8651452, 0.02438654, 0.21531618]	[-0.03266374, 0.07417766, -0.10309114, -0.01338968, 0.01648731]	1.103416	0.1
6	[0.8814277, -0.08448908, 0.872332, 0.02521412, 0.21429446]	[-0.02518997, 0.12681958, -0.07186781, -0.00827583, 0.01021719]	1.230512	0.1
7	[0.8838949, -0.09051623, 0.87892157, 0.02598571, 0.2134272]	[-0.02467204, 0.06027157, -0.06589598, -0.0077159, 0.00867264]	1.306322	0.1
8	[0.8854969, -0.104607, 0.8828858, 0.0263829, 0.21294068]	[-0.01602, 0.14090772, -0.03964229, -0.00397185, 0.00486516]	1.404978	0.0
9	[0.887363, -0.10981941, 0.887373, 0.0268438, 0.21242142]	[-0.01866129, 0.05212412, -0.04487128, -0.00460907, 0.00519266]	1.447958	0.0
10	[0.88820815, -0.11698326, 0.8891579, 0.02695693, 0.2122247]	[-0.00845115, 0.07163852, -0.01784905, -0.00113124, 0.00196704]	1.531464	0.0

6 Conclusion

In this project, I explored how different learning rates affect the training of a neural network for the prediction of gravitational wave signals. By analysing the weights, gradients, outputs, and loss over 10 steps, we observed that increasing the learning rate can lead to faster convergence, but may also risk overshooting the optimal solution.