Code 1

a basic implementation of a Wavelet Neural Network (WNN) in Python using libraries like NumPy for numerical operations and PyWavelets for wavelet transform computations.

import numpy as np

import pywt

class WaveletNeuralNetwork:

def \_\_init\_\_(self, num\_input, num\_hidden, num\_output, wavelet='db1'):

self.num\_input = num\_input

self.num\_hidden = num\_hidden

self.num\_output = num\_output

self.wavelet = wavelet

# Initialize weights and biases

self.weights\_input\_hidden = np.random.randn(num\_hidden, num\_input)

self.weights\_hidden\_output = np.random.randn(num\_output, num\_hidden)

self.bias\_hidden = np.random.randn(num\_hidden)

self.bias\_output = np.random.randn(num\_output)

def wavelet\_transform(self, input\_data):

return pywt.wavedec(input\_data, self.wavelet)

def sigmoid(self, x):

return 1 / (1 + np.exp(-x))

def forward(self, input\_data):

hidden\_input = np.dot(self.weights\_input\_hidden, input\_data) + self.bias\_hidden

hidden\_output = self.sigmoid(hidden\_input)

output\_input = np.dot(self.weights\_hidden\_output, hidden\_output) + self.bias\_output

output = self.sigmoid(output\_input)

return output

def train(self, input\_data, target\_data, learning\_rate=0.1, epochs=1000):

for \_ in range(epochs):

for i in range(len(input\_data)):

# Forward pass

input\_vector = input\_data[i]

target\_vector = target\_data[i]

hidden\_input = np.dot(self.weights\_input\_hidden, input\_vector) + self.bias\_hidden

hidden\_output = self.sigmoid(hidden\_input)

output\_input = np.dot(self.weights\_hidden\_output, hidden\_output) + self.bias\_output

output = self.sigmoid(output\_input)

# Backpropagation

output\_error = target\_vector - output

output\_delta = output\_error \* output \* (1 - output)

hidden\_error = np.dot(self.weights\_hidden\_output.T, output\_delta)

hidden\_delta = hidden\_error \* hidden\_output \* (1 - hidden\_output)

# Update weights and biases

self.weights\_hidden\_output += learning\_rate \* np.outer(output\_delta, hidden\_output)

self.bias\_output += learning\_rate \* output\_delta

self.weights\_input\_hidden += learning\_rate \* np.outer(hidden\_delta, input\_vector)

self.bias\_hidden += learning\_rate \* hidden\_delta

def predict(self, input\_data):

predictions = []

for i in range(len(input\_data)):

prediction = self.forward(input\_data[i])

predictions.append(prediction)

return np.array(predictions)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Sample input and target data

input\_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

target\_data = np.array([[0], [1], [1], [0]])

# Initialize and train the WNN

wnn = WaveletNeuralNetwork(num\_input=2, num\_hidden=4, num\_output=1)

wnn.train(input\_data, target\_data)

# Make predictions

predictions = wnn.predict(input\_data)

print("Predictions:")

for i in range(len(predictions)):

print(f"Input: {input\_data[i]}, Target: {target\_data[i]}, Prediction: {predictions[i]}")

This code demonstrates a simple implementation of a Wavelet Neural Network with one hidden layer. The **WaveletNeuralNetwork** class initializes the network with random weights and biases. It includes methods for the forward pass, training with backpropagation, and prediction. The wavelet transform is applied to the input data before feeding it into the network. This example uses the sigmoid activation function and stochastic gradient descent for optimization. You can modify it to suit your specific needs, such as using different activation functions, optimization algorithms, or network architectures.

Code 2

Here's an example implementation of a Wavelet Neural Network with Morlet wavelet and Gaussian activation functions:

import numpy as np

import pywt

class WaveletNeuralNetwork:

def \_\_init\_\_(self, num\_input, num\_hidden, num\_output, wavelet='morl'):

self.num\_input = num\_input

self.num\_hidden = num\_hidden

self.num\_output = num\_output

self.wavelet = wavelet

# Initialize weights and biases

self.weights\_input\_hidden = np.random.randn(num\_hidden, num\_input)

self.weights\_hidden\_output = np.random.randn(num\_output, num\_hidden)

self.bias\_hidden = np.random.randn(num\_hidden)

self.bias\_output = np.random.randn(num\_output)

def morlet\_wavelet\_transform(self, input\_data):

return pywt.cwt(input\_data, np.arange(1, self.num\_input + 1), self.wavelet)[0]

def gaussian(self, x, mean=0, std=1):

return np.exp(-np.power(x - mean, 2) / (2 \* np.power(std, 2)))

def forward(self, input\_data):

hidden\_input = np.dot(self.weights\_input\_hidden, input\_data) + self.bias\_hidden

hidden\_output = self.gaussian(hidden\_input)

output\_input = np.dot(self.weights\_hidden\_output, hidden\_output) + self.bias\_output

output = self.gaussian(output\_input)

return output

def train(self, input\_data, target\_data, learning\_rate=0.1, epochs=1000):

for \_ in range(epochs):

for i in range(len(input\_data)):

# Forward pass

input\_vector = input\_data[i]

target\_vector = target\_data[i]

hidden\_input = np.dot(self.weights\_input\_hidden, input\_vector) + self.bias\_hidden

hidden\_output = self.gaussian(hidden\_input)

output\_input = np.dot(self.weights\_hidden\_output, hidden\_output) + self.bias\_output

output = self.gaussian(output\_input)

# Backpropagation

output\_error = target\_vector - output

output\_delta = output\_error \* self.gaussian(output\_input, std=1) # derivative of Gaussian is itself

hidden\_error = np.dot(self.weights\_hidden\_output.T, output\_delta)

hidden\_delta = hidden\_error \* self.gaussian(hidden\_input, std=1) # derivative of Gaussian is itself

# Update weights and biases

self.weights\_hidden\_output += learning\_rate \* np.outer(output\_delta, hidden\_output)

self.bias\_output += learning\_rate \* output\_delta

self.weights\_input\_hidden += learning\_rate \* np.outer(hidden\_delta, input\_vector)

self.bias\_hidden += learning\_rate \* hidden\_delta

def predict(self, input\_data):

predictions = []

for i in range(len(input\_data)):

prediction = self.forward(input\_data[i])

predictions.append(prediction)

return np.array(predictions)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Sample input and target data

input\_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

target\_data = np.array([[0], [1], [1], [0]])

# Initialize and train the WNN

wnn = WaveletNeuralNetwork(num\_input=2, num\_hidden=4, num\_output=1)

wnn.train(input\_data, target\_data)

# Make predictions

predictions = wnn.predict(input\_data)

print("Predictions:")

for i in range(len(predictions)):

print(f"Input: {input\_data[i]}, Target: {target\_data[i]}, Prediction: {predictions[i]}")

In this implementation, I've used the Morlet wavelet transform instead of the standard wavelet transform. The activation function for both hidden and output layers is Gaussian. The rest of the implementation remains the same as the previous example. You can adjust the parameters and architecture to fit your specific problem.