Detecting Shocks Waves with Artificial Intelligence

Overview

In this project, I want to learn how to code a program to detect shock waves using artificial intelligence (AI). The problem that I want to solve is finding a way to make the code as efficient as possible while still being super sensitive and still working as intended. The reason I picked this topic is because I wanted to do something with coding and AI. I think 2,500 training iterations will be the best number of times to train the AI algorithm. The reason I think this is a good number is because 2,500 won't take a long time to run in the code and it would give it a lot of training iterations. My hypothesis for the best number of training iteration is from previous times running the AI coding and knowledge of the code. I am hoping to learn more about coding and AI from this project.

Conclusion
My conclusion is that my hypothesis is wrong. The optimal number of training iterations was 5,000, not 2,500.

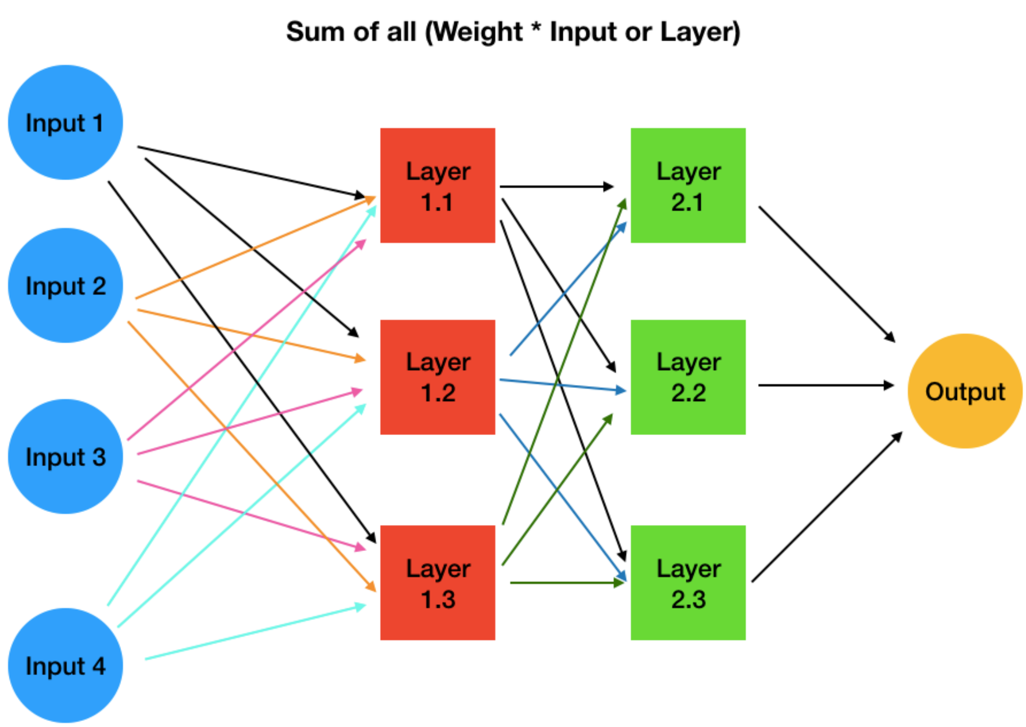


Fig: How AI works

Table 1

y x1	y x2	y x3	y x4		Smooth
1	1	0	0		0
0	0.25	0.5	1		1
0	0	1	1		0
0.75	0.5	0.25	0		1
0	0	0.25	0.75		1
0	1	1	0.5		1
0	0	1	0		0
0	1	1	0		0
1	0	0	1		0
1	1	0	1		0
0.5	0.5	0.5	0.5		1

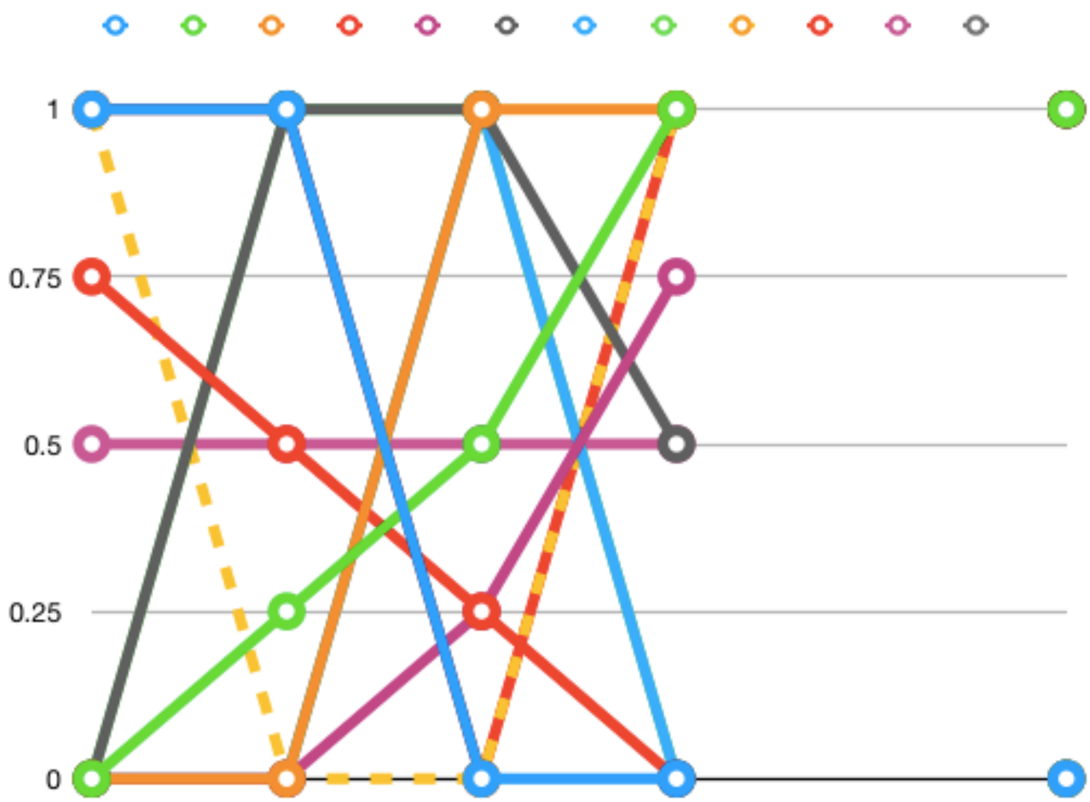
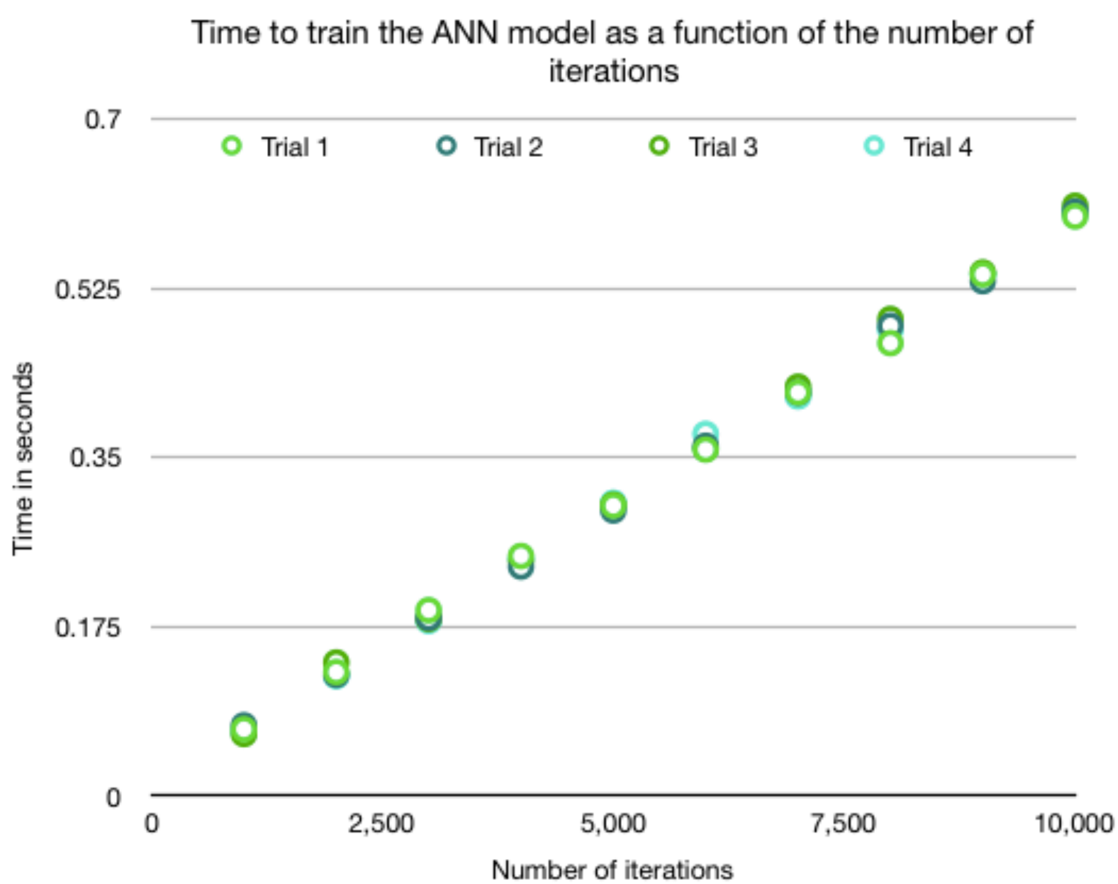


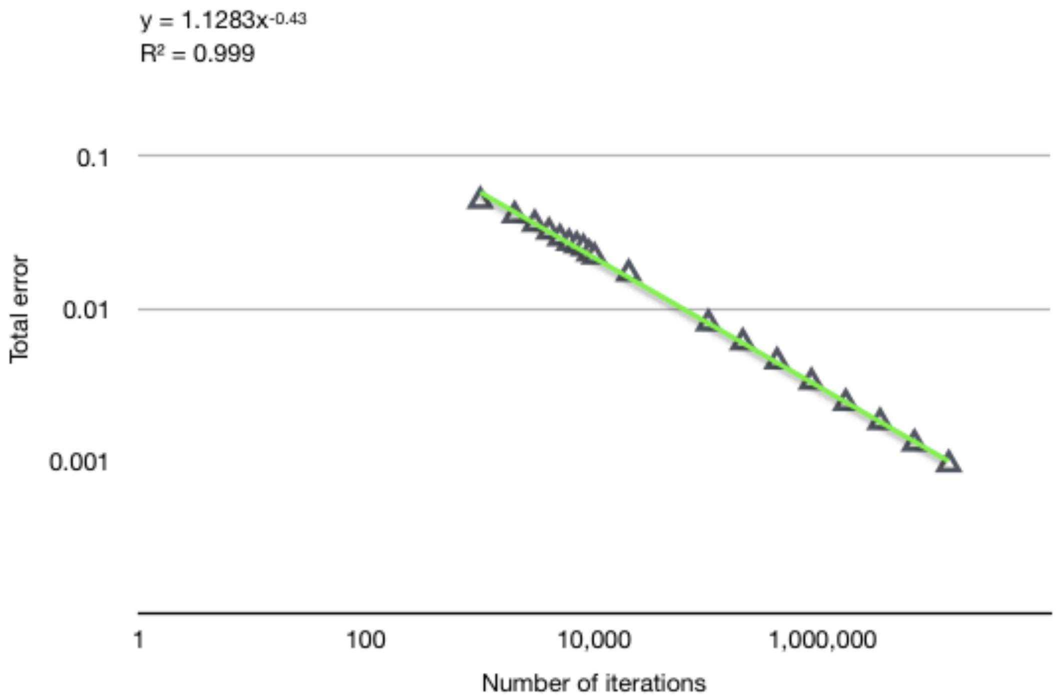
Fig: The training data for the AI code and plots of the data

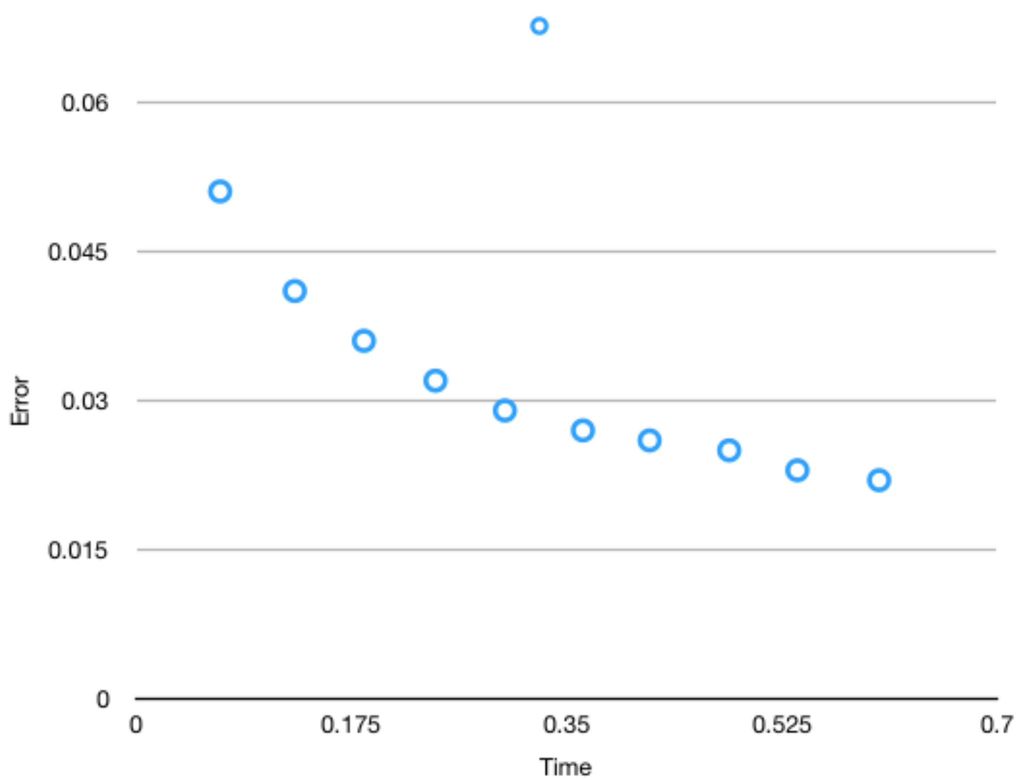
	Trial 1	Trial 2	Trial 3	Trial 4			
Number of trainings	Time	Time	Time	Time	Error at predicting	average time	Error
1,000	0.069	0.073	0.064	0.066	0.051	0.068	0.051
2,000	0.128	0.125	0.138	0.124	0.041	0.12875	0.041
3,000	0.192	0.184	0.183	0.181	0.036	0.185	0.036
4,000	0.248	0.237	0.243	0.245	0.032	0.24325	0.032
5,000	0.300	0.295	0.301	0.303	0.029	0.29975	0.029
6,000	0.358	0.362	0.359	0.374	0.027	0.36325	0.027
7,000	0.417	0.417	0.423	0.413	0.026	0.4175	0.026
8,000	0.468	0.486	0.493	0.482	0.025	0.48225	0.025
9,000	0.539	0.532	0.541	0.539	0.023	0.53775	0.023
10,000	0.599	0.604	0.610	0.604	0.022	0.60425	0.022
20,000			1.170		0.017	1.170	
100,000			6.047		0.008	6.047	
200,000			11.840		0.006	11.840	
400,000			24.608		0.0045	24.608	
800,000			49.624		0.0033	49.624	
1,600,000			93.894		0.0024	93.894	
3,200,000			189.536		0.0018	189.536	
6,400,000			373.83		0.0013	373.83	
12,800,000			764.178		0.00095	764.178	

Fig: The results from using the AI code



Total error at predicting the output values for 4 data sets as a function of the number of iterations used to train the ANN model





Code

```
#  three inputs to a N-node hidden layer to another N-node hidden layer to a single
output
#  There will be N weights going in, then N weights and then N weights going out of
hidden layer

import numpy as np
import time


def sigmoid(x):
   return 1.0/(1+ np.exp(-x))

def sigmoid_derivative(x):
   return x * (1.0 - x)

class NeuralNetwork:
   def __init__(self, x, y, w0, w1, w2):
      self.input      = x
      self.weights0   = w0
      self.weights1   = w1
      self.weights2   = w2
      self.y          = y
      self.output     = np.zeros(self.y.shape)

   def feedforward(self):
      self.layer0 = sigmoid(np.dot(self.input, self.weights0))
      self.layer1 = sigmoid(np.dot(self.layer0, self.weights1))
      self.output = sigmoid(np.dot(self.layer1, self.weights2))

   def backprop(self, eta):
      # application of the chain rule to find derivative of the loss function with respect to
weights2 and weights1

      D2 = 2*(self.y - self.output) * sigmoid_derivative(self.output)
      d_weights2 = np.dot( self.layer1.T, D2)

      D1 = np.dot(D2, self.weights2.T) * sigmoid_derivative(self.layer1)
      d_weights1 = np.dot( self.layer0.T, D1 )

      D0 = np.dot(D1, self.weights1.T) * sigmoid_derivative(self.layer0)
      d_weights0 = np.dot( self.input.T, D0 )
```

```
        # update the weights with the derivative (slope) of the loss function
        self.weights0 += eta*d_weights0
        self.weights1 += eta*d_weights1  # add \eta in front
        self.weights2 += eta*d_weights2




# This is the main program
if __name__ == "__main__":


    # the input values for training
    X = np.loadtxt("X.txt", delimiter=",")
    print(X)


    # the desired outputs
    Yrow = np.loadtxt("Y.txt")
    Y = Yrow.reshape(X.shape[0],1)
    print(Y)

    # initialize the weights
    size = 3  # the number of nodes in the hidden layer

    # to generate input values for the weights
    #W0 = np.random.rand(X.shape[1],size)
    #W1 = np.random.rand(size,size)
    #W2 = np.random.rand(size,1)


    W0 = np.loadtxt("w0.txt", delimiter=",")
    W1 = np.loadtxt("w1.txt", delimiter=",")
    readW2 = np.loadtxt("w2.txt", delimiter=",")
    W2 = readW2.reshape(size,1)


    # create the neural network
    nn = NeuralNetwork(X,Y,W0,W1,W2)

    eta = 1.0

    # calculate start time
    start_time = time.time()
```

```
# train the neural network
for i in range(1000):
    nn.feedforward()
    nn.backprop(eta)


# calculate end time
end_time = time.time()

# list the outputs
print(nn.output)

# time to run the code
print("Elapsed time was %g seconds" % (end_time - start_time))


#  write weights to a file
np.savetxt("w0out.txt", W0, fmt="%3.16f", delimiter=",")
np.savetxt("w1out.txt", W1, fmt="%3.16f", delimiter=",")
np.savetxt("w2out.txt", W2, fmt="%3.16f", delimiter=",")



# use the neural network on data
dataX = np.array([[1,0.7,.3,0],
                  [1,0.8,0,0],
                  [0.2,0.4,0.8,1.0],
                  [0,0,0.9,1.0]])

# we should get these values
dataY = np.array([[1],
                  [0],
                  [1],
                  [0]])
datann = NeuralNetwork(dataX,dataY,W0,W1,W2)
datann.feedforward()
print(datann.output)


Error = np.abs(datann.output-dataY)
print('error =')
print(Error)
print(np.sum(Error))
```

References

1. ai.google
2. https://www.instructables.com/id/Build-Your-Own-AI-Artificial-Intellige nce-Assistan/
3. images.google
4. youtube.com
5. https://www.udacity.com/course/ai-programming-python-nanodegree--nd089