```
import sys
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt

def linear(X_input, weights):
    """
    out = Xw
    X should be (rows, columns) and w should match (columns, hidden units)
    """
    a = np.dot(X_input, weights)
    return a

def sigmoid(hidden_input):
    a = 1 / (1 + np.exp(-hidden_input))
    return a

def MSE(y, yhat):
    return np.mean((y - yhat)**2)


class NeuralNetwork:

    def __init__(self, X, y, hidden_units = 9, learning_rate = 0.1):
        """
        Two hidden layer neural network for regression
            Hidden Layer 1: Logistic Activation
            Hidden Layer 2: Linear Activation (for regression)
        """
        self.X = X
        self.y = y
        self.learning_rate = learning_rate
        self.hidden_units = hidden_units
        np.random.seed(123)
        self.weights1 = np.random.normal(loc=0, scale=1,
                            size=(X.shape[1], hidden_units)) # (input units, hidden units)
        self.weights2 = np.random.normal(loc=0, scale=1,
                            size=(hidden_units, 1)) # for now there will only be one output


    def forward(self, X = None, verbose=False):
        if isinstance(X, np.ndarray):
            hidden_output1 = linear(X, self.weights1)
            hidden_output1 = sigmoid(hidden_output1) # apply the nonlinear transformation
            hidden_output2 = linear(hidden_output1, self.weights2)

            if verbose:
                print("hidden output: ", hidden_output1.round(3))
                print("output: ", hidden_output2.round(3))

            return hidden_output2, hidden_output1
        else:
            raise ValueError("need a vector in the forward function")


    def backpropogation(self, hidden_output1, hidden_output2, x, target):
        output_error = (target - hidden_output2) #  -(target - y); 20 x 1

        # update hidden layer weights
        hidden_error = np.dot(self.weights2, output_error) # neccessary for hidden layer upd
        update1 = hidden_error * hidden_output1 * (1 - hidden_output1) # logistic output fro
        update1 = update1*x[:, None] # multiply the input units as is part of the logistic d

        # update output layer weights; linear not logistic
        update2 = output_error * hidden_output1 # use hidden layer outputs to update output
        return update2[:, None], update1

    def gradient_descent(self, delta_weights1, delta_weights2):
        self.weights1 += self.learning_rate * delta_weights1 # hidden layer weights
        self.weights2 += self.learning_rate * delta_weights2 # output layer weights
        return None


    def train(self, n_epochs = 15):
        n_records = self.X.shape[0]
        for _ in range(n_epochs):
            delta_weights1 = np.zeros_like(self.weights1)
            delta_weights2 = np.zeros_like(self.weights2)
            for x, y in zip(self.X, self.y):
                hidden_output2, hidden_output1 = self.forward(X=x, verbose=False)
                update2, update1 = self.backpropogation(hidden_output1, hidden_output2, x=x,
                delta_weights1 += update1 / n_records
                delta_weights2 += update2 / n_records
            self.gradient_descent(delta_weights1, delta_weights2)
            yhat, _ = self.forward(X=self.X)
            mse_stat = MSE(y = self.y, yhat = yhat)
            #print("MSE: ", mse_stat)
        return yhat
```

```python
def example_data(rows = 20, columns = 3):
    #np.random.seed(123)
    X = np.random.normal(size=(rows, columns))
    X[:, 0] = np.linspace(start=-10, stop=10, num=rows)**4
    X[:, 1] = np.linspace(start=-10, stop=10, num=rows)
    X[:, 2] = np.linspace(start=-10, stop=10, num=rows)**3
    y = 1.5 * X[:, 0] + X[:, 1] + 2 * X[:, 2]
    y = y[:, None]
    ## scale the values ##
    X = X / 1000
    y = (y - 68) / 500
    return (X, y)


def forward_run_example():
    X, y = example_data()
    model = NeuralNetwork(X, y)
    print("Forward Run")
    for x, _ in zip(X,y):
        hidden_output2, hidden_output1 = model.forward(X=x) # returns (final_output, hidden_
        #print(x,target)
    return model, hidden_output2, hidden_output1, X, y

#forward_run_example()



def backprop_run_example():
    model, hidden_output2, hidden_output1, X, y = forward_run_example()
    print("Backward Run")
    for (x, target) in zip(X, y):
        update2, update1 = model.backpropogation(hidden_output1, hidden_output2, x, target)
    return model, update2, update1


#backprop_run_example()

def gradient_descent_example():
    model, update2, update1 = backprop_run_example()
    model.gradient_descent(update1, update2)


#gradient_descent_example()
```

```python
def run_example():
    X, y = example_data()
    model = NeuralNetwork(X, y, hidden_units=100, learning_rate=0.004)
    yhat = model.train(n_epochs=1000)
    yhat = yhat*500 + 68
    y = y*500 + 68
    mse_stat = MSE(y = y, yhat = yhat)
    #print("Final MSE (not scaled): ", mse_stat)
    for i in range(3):
        plt.title("X[:, {}] and y".format(i))
        plt.scatter(X[:, i], y=y, marker='o')
        plt.scatter(X[:, i], y=yhat, marker='x')
        plt.show()

    return model

run_example()
```