```
import numpy as np

class DataHandler:
  def __init__(self, X, y):
    self.X = X
    self.y = y

  def clean_data(self):
    self.X = self.X / np.max(self.X, axis=0)
    return self.X, self.y


class NeuralNetwork:
  def __init__(self, input_size, hidden_size,
  output_size, learning_rate=0.1):
    self.lr = learning_rate
    self.W1 = np.random.randn(input_size,
    hidden_size)
    self.b1 = np.zeros((1, hidden_size))
    self.W2 = np.random.randn(hidden_size,
    output_size)
    self.b2 = np.zeros((1, output_size))

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

  def sigmoid_derivative(self, x):
    return x * (1 - x)

  def forward(self, X):
    self.z1 = np.dot(X, self.W1) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    self.output = self.sigmoid(self.z2)
    return self.output

  def backward(self, X, y, output):
    error = y - output
    d_output = error *
    self.sigmoid_derivative(output)
    error_hidden = d_output.dot(self.W2.T)
    d_hidden = error_hidden *
    self.sigmoid_derivative(self.a1)
    self.W2 += self.a1.T.dot(d_output) * self.lr
    self.b2 += np.sum(d_output, axis=0,
    keepdims=True) * self.lr
    self.W1 += X.T.dot(d_hidden) * self.lr
    self.b1 += np.sum(d_hidden, axis=0,
    keepdims=True) * self.lr

  def train(self, X, y, epochs=10000):
    for _ in range(epochs):
      output = self.forward(X)
      self.backward(X, y, output)


class Predictor:
  def __init__(self, model):
    self.model = model

  def predict(self, X):
    output = self.model.forward(X)
    return (output > 0.5).astype(int)

  def accuracy(self, X, y):
    preds = self.predict(X)
    correct = np.sum(preds == y)
    return correct / len(y) * 100
```

```
logic_gates = {
  "AND": {
    "X": np.array([[0,0],[0,1],[1,0],[1,1]]),
    "y": np.array([[0],[0],[0],[1]])
  },
  "OR": {
    "X": np.array([[0,0],[0,1],[1,0],[1,1]]),
    "y": np.array([[0],[1],[1],[1]])
  },
  "XOR": {
    "X": np.array([[0,0],[0,1],[1,0],[1,1]]),
    "y": np.array([[0],[1],[1],[0]])
  },
  "NAND": {
    "X": np.array([[0,0],[0,1],[1,0],[1,1]]),
    "y": np.array([[1],[1],[1],[0]])
  },
  "NOR": {
    "X": np.array([[0,0],[0,1],[1,0],[1,1]]),
    "y": np.array([[1],[0],[0],[0]])
  },
  "XNOR": {
    "X": np.array([[0,0],[0,1],[1,0],[1,1]]),
    "y": np.array([[1],[0],[0],[1]])
  },
  "NOT": {
    "X": np.array([[0],[1]]),
    "y": np.array([[1],[0]])
  }
}
```

```
if __name__ == "__main__":
  gate = input("Enter gate (AND, OR, XOR, NAND,
  NOR, XNOR, NOT): ").upper()

  if gate not in logic_gates:
    print("Invalid gate name!")
  else:
    X = logic_gates[gate]["X"]
    y = logic_gates[gate]["y"]

    data = DataHandler(X, y)
    X_clean, y_clean = data.clean_data()

    input_size = X_clean.shape[1]
    hidden_size = 2
    output_size = 1
    learning_rate = 0.1
    epochs = 10000

    nn = NeuralNetwork(input_size, hidden_size,
    output_size, learning_rate=learning_rate)
    nn.train(X_clean, y_clean, epochs=epochs)

    predictor = Predictor(nn)
    predictions = predictor.predict(X_clean)
    acc = predictor.accuracy(X_clean, y_clean)

    print(f"\nLogic Gate: {gate}")
    print(f"Learning Rate: {learning_rate}")
    print(f"Epochs Trained: {epochs}")
    print("Input:\n", X)
    print("Expected:\n", y)
    print("Predictions:\n", predictions)
    print(f"Accuracy: {acc:.2f}%")
```