Learn From Scratch: Backpropagation Neural Networks Using Python GUI, MariaDB, and CRUD

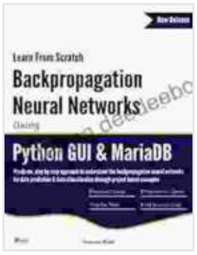
Backpropagation neural networks are powerful machine learning models that can solve a wide range of problems, from image classification to natural language processing. However, understanding how they work can be a challenge. In this article, we'll build a simple backpropagation neural network in Python with a graphical user interface (GUI) and MariaDB database connectivity. This hands-on approach will help you gain a deeper understanding of the concepts behind neural networks.

Prerequisites

- Basic understanding of Python programming
- Access to a Python development environment
- MariaDB database installed and configured
- Knowledge of CRUD (Create, Read, Update, Delete) operations in MariaDB

Step 1: Create a Python GUI

We'll use the PyQt5 library to create a simple GUI for our neural network. Start by creating a new Python file and adding the following code:



File size	: 63051 KB
Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 862 pages
Lending	: Enabled
X-Ray for textbooks	: Enabled
Paperback	: 328 pages
Item Weight	: 1.24 pounds
Dimensions	: 6 x 0.83 x 9 inches

python import sys from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout from PyQt5.QtGui import QPixmap, QImage

class NeuralNetworkGUI(QWidget):

def __init__(self): super().__init__()

self.initUI()

def initUI(self): # Window settings self.setGeometry(300, 300, 500, 500) self.setWindowTitle('Neural Network GUI')

# Layout self.main_layout = QVBoxLayout() self.setLayout(self.main_layout)

# Image label self.image_label = QLabel() self.image_label.setFixedSize(200, 200) self.main_layout.addWidget(self.image_label)

```
# Buttons self.train_button = QPushButton('Train')
self.train_button.clicked.connect(self.train_network)
self.main_layout.addWidget(self.train_button)

self.test_button = QPushButton('Test')
self.test_button.clicked.connect(self.test_network)
self.main_layout.addWidget(self.test_button)
```

Step 2: Load the Image Data

Next, we need to load the image data that we want to train our neural network on. In this example, we'll use the MNIST dataset, which contains handwritten digits.

```
python import PIL.Image import numpy as np

def load_data(): # Load images from MNIST dataset train_images =
PIL.Image.open('train-images.idx3-ubyte') train_images =
np.array(train_images).reshape((60000, 28, 28))

train_labels = PIL.Image.open('train-labels.idx1-ubyte') train_labels =
np.array(train_labels).reshape((60000,))

test_images = PIL.Image.open('test-images.idx3-ubyte') test_images =
np.array(test_images).reshape((10000, 28, 28))

test_labels = PIL.Image.open('test-labels.idx1-ubyte') test_labels =
np.array(test_labels).reshape((10000,))

return train_images, train_labels, test_images, test_labels
```

Step 3: Create the Neural Network Model

Now we can create the neural network model. We'll use a feedforward network with one hidden layer.

python import numpy as np

class NeuralNetwork:

def __init__(self, input_size, hidden_size, output_size): # Initialize weights and biases self.weights1 = np.random.randn(input_size, hidden_size) / np.sqrt(input_size) self.biases1 = np.zeros((1, hidden_size)) self.weights2 = np.random.randn(hidden_size, output_size) / np.sqrt(hidden_size) self.biases2 = np.zeros((1, output_size))

def forward(self, x): # Forward pass z1 = np.dot(x, self.weights1) + self.biases1 a1 = np.tanh(z1) z2 = np.dot(a1, self.weights2) + self.biases2 a2 = np.tanh(z2) return a2

def backward(self, x

Dimensions	: 6 x 0.83 x 9 inches