In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.utils import to_categorical
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# ReLU activation function
def relu(x):
    return np.maximum(0, x)

# Derivative of ReLU function
def relu_derivative(x):
    return np.where(x > 0, 1, 0)

# Softmax activation function
def softmax(x):
    exps = np.exp(x - np.max(x, axis=1, keepdims=True))  # Stability improve
    return exps / np.sum(exps, axis=1, keepdims=True)

# Categorical Cross-Entropy Loss function
def categorical_crossentropy(predictions, labels):
    return -np.mean(np.sum(labels * np.log(predictions + 1e-8), axis=1))

# Accuracy function
def accuracy(y_true, y_pred):
    correct = np.argmax(y_true, axis=1) == np.argmax(y_pred, axis=1)
    return np.mean(correct)
```

In [3]:
```
# Load and preprocess the MNIST dataset
def load_mnist():
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    # Flatten images into 784-dimensional vectors
    X_train = X_train.reshape(X_train.shape[0], -1) / 255.0  # Normalize to
    X_test = X_test.reshape(X_test.shape[0], -1) / 255.0

    # One-hot encode the labels
    y_train = to_categorical(y_train, 10)
    y_test = to_categorical(y_test, 10)

    return X_train, y_train, X_test, y_test
```

In [4]:
```
# Initialize parameters
def initialize_weights(input_size, hidden_size, output_size):
    W1 = np.random.randn(input_size, hidden_size) * 0.01  # Weights for the
    W2 = np.random.randn(hidden_size, output_size) * 0.01  # Weights for the
    return W1, W2
```

In [6]:
```
# Main training function
def train_neural_network(X_train, y_train, hidden_size=128, output_size=10,
    input_size = X_train.shape[1]  # 784 input neurons (28x28 pixels)
```

```python
    N = X_train.shape[0]  # Number of training examples

    # Initialize weights
    W1, W2 = initialize_weights(input_size, hidden_size, output_size)

    # DataFrame to store loss and accuracy
    results = pd.DataFrame(columns=["loss", "accuracy"])

    for itr in range(iterations):
        # Feedforward propagation
        Z1 = np.dot(X_train, W1)  # First layer pre-activation
        A1 = relu(Z1)  # First layer activation (ReLU)

        Z2 = np.dot(A1, W2)  # Second layer pre-activation (output)
        A2 = softmax(Z2)  # Second layer activation (output predictions usir

        # Calculate loss (Categorical Cross-Entropy)
        loss = categorical_crossentropy(A2, y_train)
        acc = accuracy(y_train, A2)

        # Store loss and accuracy for each iteration
        new_row = pd.DataFrame({"loss": [loss], "accuracy": [acc]})
        results = pd.concat([results, new_row], ignore_index=True)

        # Backpropagation
        # Gradient for softmax output layer
        E1 = A2 - y_train  # Error at output layer
        dW2 = np.dot(A1.T, E1) / N  # Gradient for W2

        # Error propagated to hidden layer
        E2 = np.dot(E1, W2.T)  # Backpropagated error
        dZ1 = E2 * relu_derivative(Z1)  # Derivative of error w.r.t. Z1 (for
        dW1 = np.dot(X_train.T, dZ1) / N  # Gradient for W1

        # Update weights
        W2 = W2 - learning_rate * dW2  # Update weights for W2
        W1 = W1 - learning_rate * dW1  # Update weights for W1

    return results, W1, W2
```

In [7]:
```python
# Load the dataset
X_train, y_train, X_test, y_test = load_mnist()
```

In [8]:
```python
# Train the neural network
results, W1, W2 = train_neural_network(X_train, y_train)
```

In [ ]:
```python
# After training, print final results
print("Final loss:", results['loss'].iloc[-1])
print("Final accuracy:", results['accuracy'].iloc[-1])
```

In [ ]:
```python
# Plot Loss (Categorical Cross-Entropy)
results.loss.plot(title="Categorical Cross-Entropy Loss")
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```

```python
# Plot Accuracy
results.accuracy.plot(title="Accuracy")
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.show()
```

This notebook was converted to PDF with convert.ploomber.io