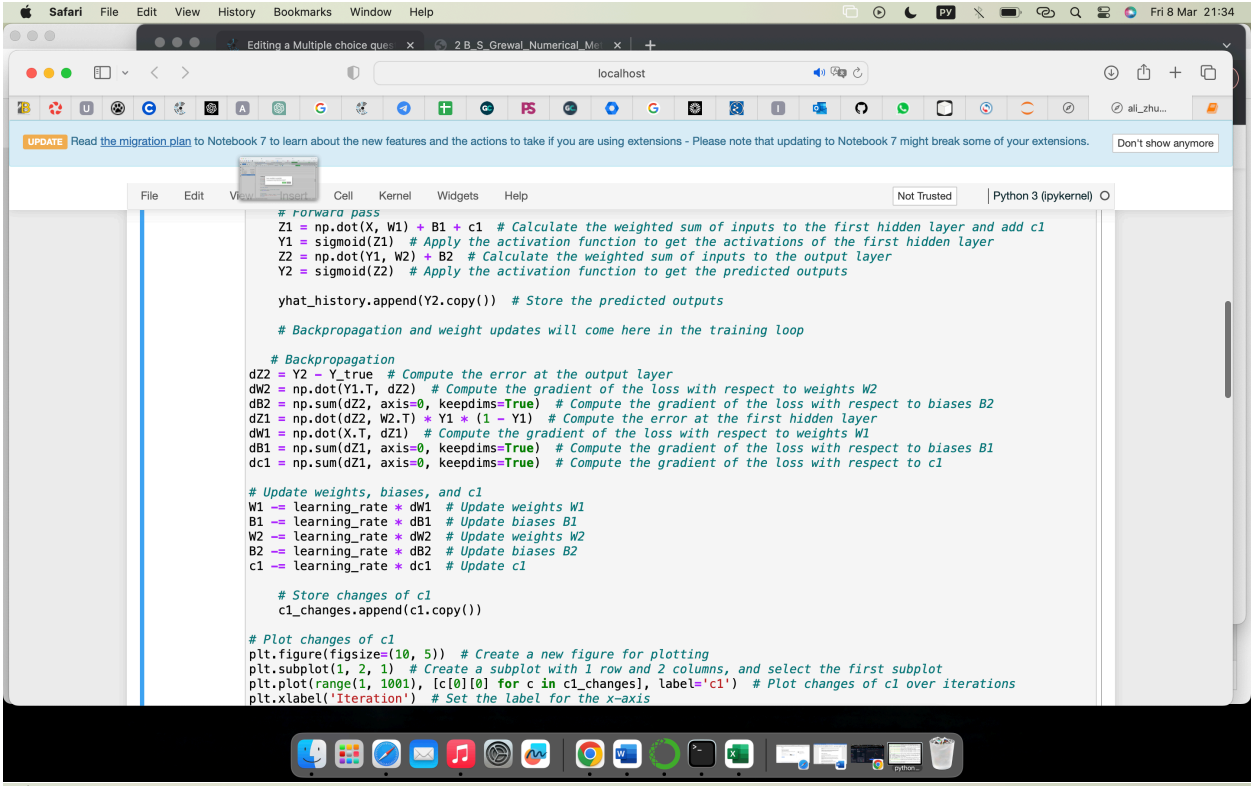
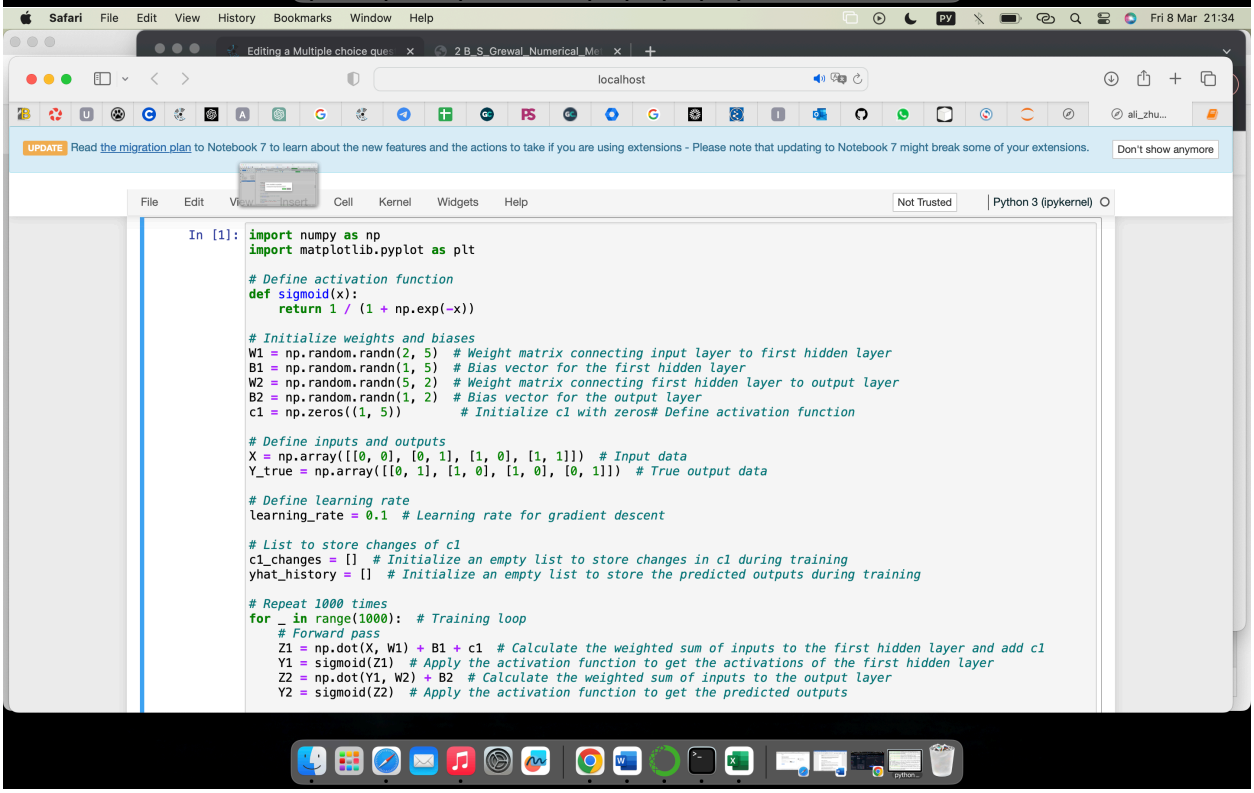
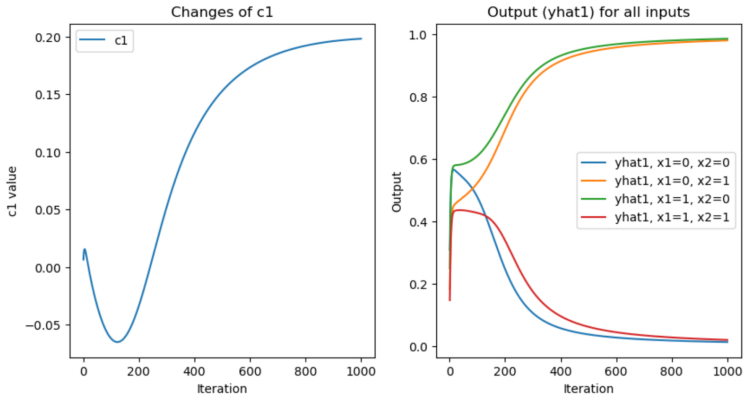


```
        # Forward pass
        Z1 = np.dot(X, W1) + B1 + c1  # Calculate the weighted sum of inputs to the first hidden layer and add c1
        Y1 = sigmoid(Z1)  # Apply the activation function to get the activations of the first hidden layer
        Z2 = np.dot(Y1, W2) + B2  # Calculate the weighted sum of inputs to the output layer
        Y2 = sigmoid(Z2)  # Apply the activation function to get the predicted outputs

        yhat_history.append(Y2.copy())  # Store the predicted outputs

        # Backpropagation and weight updates will come here in the training loop

       # Backpropagation
    dZ2 = Y2 - Y_true  # Compute the error at the output layer
    dW2 = np.dot(Y1.T, dZ2)  # Compute the gradient of the loss with respect to weights W2
    dB2 = np.sum(dZ2, axis=0, keepdims=True)  # Compute the gradient of the loss with respect to biases B2
    dZ1 = np.dot(dZ2, W2.T) * Y1 * (1 - Y1)  # Compute the error at the first hidden layer
    dW1 = np.dot(X.T, dZ1)  # Compute the gradient of the loss with respect to weights W1
    dB1 = np.sum(dZ1, axis=0, keepdims=True)  # Compute the gradient of the loss with respect to biases B1
    dc1 = np.sum(dZ1, axis=0, keepdims=True)  # Compute the gradient of the loss with respect to c1

    # Update weights, biases, and c1
    W1 -= learning_rate * dW1  # Update weights W1
    B1 -= learning_rate * dB1  # Update biases B1
    W2 -= learning_rate * dW2  # Update weights W2
    B2 -= learning_rate * dB2  # Update biases B2
    c1 -= learning_rate * dc1  # Update c1

        # Store changes of c1
        c1_changes.append(c1.copy())

    # Plot changes of c1
    plt.figure(figsize=(10, 5))  # Create a new figure for plotting
    plt.subplot(1, 2, 1)  # Create a subplot with 1 row and 2 columns, and select the first subplot
    plt.plot(range(1, 1001), [c[0][0] for c in c1_changes], label='c1')  # Plot changes of c1 over iterations
    plt.xlabel('Iteration')  # Set the label for the x-axis
```



```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        # Define activation function
        def sigmoid(x):
            return 1 / (1 + np.exp(-x))

        # Initialize weights and biases
        W1 = np.random.randn(2, 5)  # Weight matrix connecting input layer to first hidden layer
        B1 = np.random.randn(1, 5)  # Bias vector for the first hidden layer
        W2 = np.random.randn(5, 2)  # Weight matrix connecting first hidden layer to output layer
        B2 = np.random.randn(1, 2)  # Bias vector for the output layer
        c1 = np.zeros((1, 5))        # Initialize c1 with zeros# Define activation function

        # Define inputs and outputs
        X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input data
        Y_true = np.array([[0, 1], [1, 0], [1, 0], [0, 1]])  # True output data

        # Define learning rate
        learning_rate = 0.1  # Learning rate for gradient descent

        # List to store changes of c1
        c1_changes = []  # Initialize an empty list to store changes in c1 during training
        yhat_history = []  # Initialize an empty list to store the predicted outputs during training

        # Repeat 1000 times
        for _ in range(1000):  # Training loop
            # Forward pass
            Z1 = np.dot(X, W1) + B1 + c1  # Calculate the weighted sum of inputs to the first hidden layer and add c1
            Y1 = sigmoid(Z1)  # Apply the activation function to get the activations of the first hidden layer
            Z2 = np.dot(Y1, W2) + B2  # Calculate the weighted sum of inputs to the output layer
            Y2 = sigmoid(Z2)  # Apply the activation function to get the predicted outputs
```

```
plt.tight_layout()  # Adjust subplot parameters to give specified padding
plt.show()  # Show the plot
```

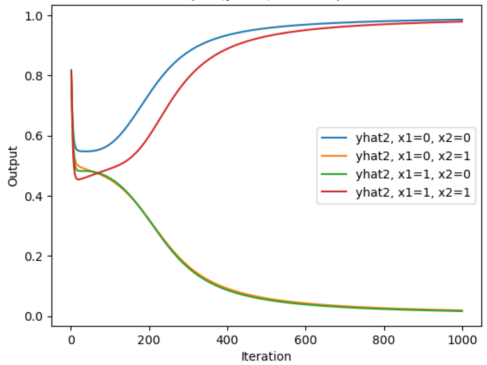



Output (yhat2) for all inputs

Iteration

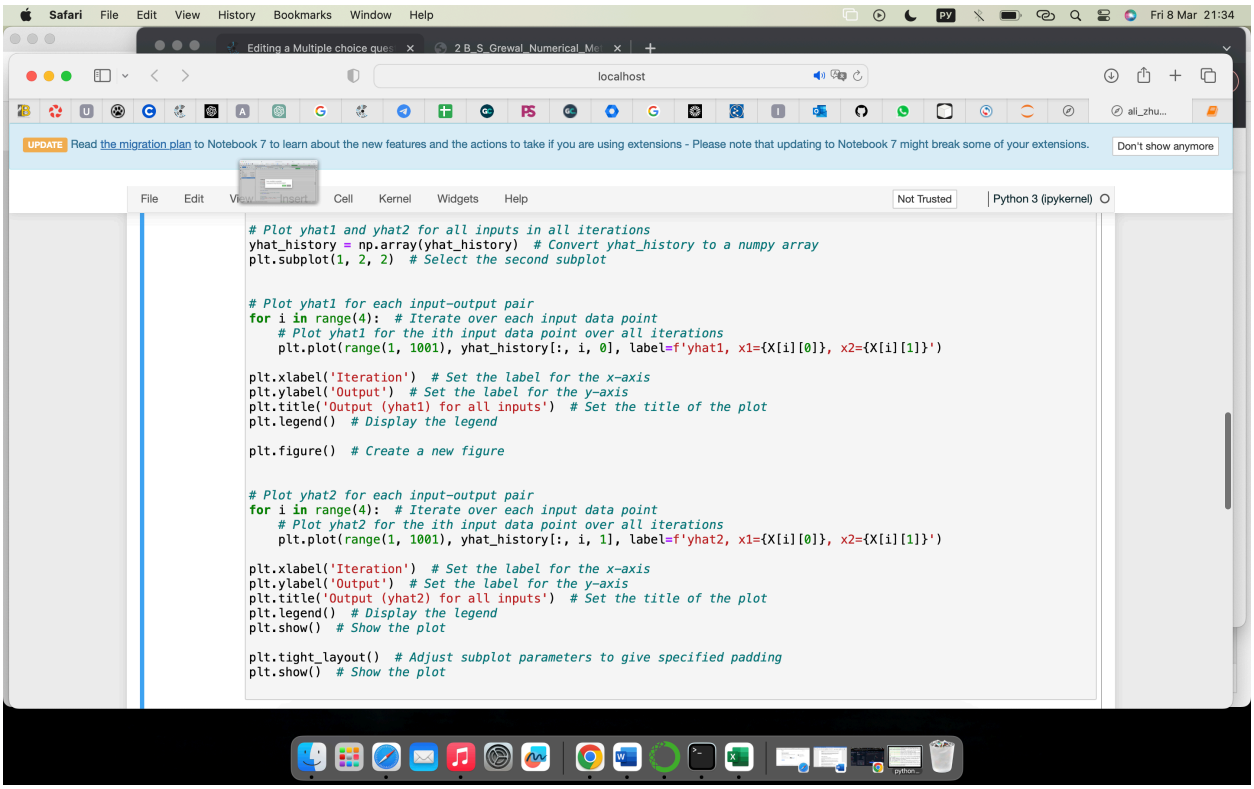
Iteration

Output (yhat2) for all inputs




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:


Safari File Edit View History Bookmarks Window Help
Fri 8 Mar 21:34
localhost
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
# Plot yhat1 and yhat2 for all inputs in all iterations
yhat_history = np.array(yhat_history)  # Convert yhat_history to a numpy array
plt.subplot(1, 2, 2)  # Select the second subplot
# Plot yhat1 for each input-output pair
for i in range(4):  # Iterate over each input data point
# Plot yhat1 for the ith input data point over all iterations
plt.plot(range(1, 1001), yhat_history[:, i, 0], label=f'yhat1, x1={X[i][0]}, x2={X[i][1]}')
plt.xlabel('Iteration')  # Set the label for the x-axis
plt.ylabel('Output')  # Set the label for the y-axis
plt.title('Output (yhat1) for all inputs')  # Set the title of the plot
plt.legend()  # Display the legend
plt.figure()  # Create a new figure
# Plot yhat2 for each input-output pair
for i in range(4):  # Iterate over each input data point
# Plot yhat2 for the ith input data point over all iterations
plt.plot(range(1, 1001), yhat_history[:, i, 1], label=f'yhat2, x1={X[i][0]}, x2={X[i][1]}')
plt.xlabel('Iteration')  # Set the label for the x-axis
plt.ylabel('Output')  # Set the label for the y-axis
plt.title('Output (yhat2) for all inputs')  # Set the title of the plot
plt.legend()  # Display the legend
plt.show()  # Show the plot
plt.tight_layout()  # Adjust subplot parameters to give specified padding
plt.show()  # Show the plot