```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
dataset_path = '/content/gdrive/My Drive/ME5201_assignment/Training Dataset ME5201.xlsx'
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
# Load data from Excel
data = pd.read_excel(dataset_path)

# Extract features (X1, X2) and target variable (y)
X = data[['Density of Fluid (kg/m3)', 'Bulk Modulus of Fluid (Pa)']].values
y = data['Sound wave speed in fluid (m/s)'].values
```

```
# Split the data into train, validation, and test sets (70-20-10)
num_samples = X.shape[0]
num_train = int(0.7 * num_samples)
num_val = int(0.2 * num_samples)
num_test = num_samples - num_train - num_val
```

```
X_train, y_train = X[:num_train], y[:num_train]
X_val, y_val = X[num_train:num_train + num_val], y[num_train:num_train + num_val]
X_test, y_test = X[num_train + num_val:], y[num_train + num_val:]
```

```
# Normalize the data (beneficial for neural networks)
mean_X, std_X = np.mean(X_train, axis=0), np.std(X_train, axis=0)
X_train = (X_train - mean_X) / std_X
X_val = (X_val - mean_X) / std_X
X_test = (X_test - mean_X) / std_X
```

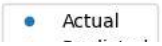
```
# Define hyperparameters
input_size = X_train.shape[1]
hidden_size = 300
output_size = 1
learning_rate = 0.001
epochs = 1000
gradient_clip = 5.0  # Set a threshold for gradient clipping
```

```
# Initialize weights and biases
np.random.seed(42)
weights_input_hidden = np.random.randn(input_size, hidden_size)
bias_hidden = np.zeros(hidden_size)
weights_hidden_output = np.random.randn(hidden_size, output_size)
bias_output = np.zeros(output_size)
```

```
# Training loop
for epoch in range(epochs):
    # Forward pass
    hidden_input = np.dot(X_train, weights_input_hidden) + bias_hidden
    hidden_output = np.maximum(0, hidden_input)  # ReLU activation
    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    y_pred = final_input

    # Compute loss (mean squared error)
    loss = np.mean((y_pred - y_train.reshape(-1, 1))**2)

    # Backward pass
    grad_y_pred = 2 * (y_pred - y_train.reshape(-1, 1)) / num_train
    grad_weights_hidden_output = np.dot(hidden_output.T, grad_y_pred)
    grad_bias_output = np.sum(grad_y_pred, axis=0)
    grad_hidden_output = np.dot(grad_y_pred, weights_hidden_output.T)
    grad_hidden_input = grad_hidden_output * (hidden_input > 0)  # ReLU gradient

    # Gradient clipping
    grad_norm = np.linalg.norm(grad_hidden_input)
    if grad_norm > gradient_clip:
        grad_hidden_input = gradient_clip * grad_hidden_input / grad_norm

    grad_weights_input_hidden = np.dot(X_train.T, grad_hidden_input)
    grad_bias_hidden = np.sum(grad_hidden_input, axis=0)

    # Update weights and biases
    weights_hidden_output -= learning_rate * grad_weights_hidden_output
    bias_output -= learning_rate * grad_bias_output
    weights_input_hidden -= learning_rate * grad_weights_input_hidden
    bias_hidden -= learning_rate * grad_bias_hidden


# Predictions on validation set
hidden_input_val = np.dot(X_val, weights_input_hidden) + bias_hidden
hidden_output_val = np.maximum(0, hidden_input_val)
final_input_val = np.dot(hidden_output_val, weights_hidden_output) + bias_output
y_val_pred = final_input_val.flatten()


# Calculate mean squared error and r2 score
mse = np.mean((y_val_pred - y_val)**2)
ssr = np.sum((y_val_pred - np.mean(y_val))**2)
sst = np.sum((y_val - np.mean(y_val))**2)
r2 = ssr / sst


print(f'Mean Squared Error on Validation Set: {mse}')
print(f'R^2 Score on Validation Set: {r2}')
```

```
Mean Squared Error on Validation Set: 14485.633144362357
R^2 Score on Validation Set: 0.8634050226654129
```

```
# Plot predicted vs actual in 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_val[:, 0], X_val[:, 1], y_val, label='Actual')
ax.scatter(X_val[:, 0], X_val[:, 1], y_val_pred, label='Predicted', marker='^')
ax.set_xlabel('Density of Fluid (kg/m3)')
ax.set_ylabel('Bulk Modulus of Fluid (Pa)')
ax.set_zlabel('Sound wave speed in fluid (m/s)')
ax.legend()
plt.show()
```




```
# Manually check predictions
def predict_manual(x1, x2):
    # Normalize input (assuming you normalized during training)
    x_normalized = (np.array([x1, x2]) - mean_X) / std_X

    # Forward pass
    hidden_input = np.dot(x_normalized, weights_input_hidden) + bias_hidden
    hidden_output = np.maximum(0, hidden_input)  # ReLU activation
    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    y_pred = final_input.flatten()

    return y_pred[0]
```



```
# Get user input for X1 and X2
x1_value = float(input("Enter the value for X1: "))
x2_value = float(input("Enter the value for X2: "))


# Predict Y manually
predicted_y = predict_manual(x1_value, x2_value)
actual_y = data.loc[(data['Density of Fluid (kg/m3)'] == x1_value) & (data['Bulk Modulus of Fluid (Pa)'] == x2_value), 'Sound wave


# Print the results
print(f"Manual Prediction for X1={x1_value}, X2={x2_value}: {predicted_y:.4f}")
print(f"Actual Y from the dataset: {actual_y:.4f}")
```

```
Enter the value for X1: 1.57e3
Enter the value for X2: 2.21e9
Manual Prediction for X1=1570.0, X2=2210000000.0: 1109.4904
Actual Y from the dataset: 1111.1100
```

```
# Print the equation
print("Equation of the fit:")
print(f"Y = {bias_output[0]:.4f}", end=' ')
```

```
Equation of the fit:
Y = 46.1131
```

```
# For each input feature and its corresponding weight in the hidden layer
for i in range(input_size):
    weight = weights_input_hidden[i, 0]
    print(f"+ {weight:.4f} * X{i + 1}", end=' ')
```

```
+ 0.5425 * X1 + -0.6581 * X2
```

```
# Bias term for the hidden layer
print(f"+ {bias_hidden[0]:.4f}", end=' ')
```

```
+ 0.4985
```

```
# For the output layer
print(f"+ {weights_hidden_output[0, 0]:.4f} * ReLU(hidden_input) + {bias_output[0]:.4f}")
```

```
+ 3.5408 * ReLU(hidden_input) + 46.1131
```