Here's a step-by-step guide to help you complete your deep learning homework, based on the provided documents:

**Step 1: Setup Your Environment**

You will need to install and set up the necessary environment for Python-based deep learning:

1. **Install Anaconda**: Follow the instructions from the guideline to install Anaconda, which provides the necessary tools such as Spyder or Jupyter Notebook.
   * [Anaconda Installation Guide](https://docs.anaconda.com/anaconda/install/windows/)​(2024\_HW1\_guideline)
2. **Install Required Libraries**:
   * **NumPy** for numerical operations.
   * **Matplotlib** for plotting learning curves and other visualizations.
   * **Pandas** for handling data (loading and preprocessing). You can install these by running:

bash

Copy code

conda install numpy pandas matplotlib

**Step 2: Work on the Regression Task**

1. **Load and Preprocess the Dataset**:
   * **Load the energy efficiency dataset** using Pandas (2024\_energy\_efficiency\_data.csv).
   * Shuffle the data and split it into training (75%) and testing (25%) sets.
   * **One-hot encode** categorical features such as orientation and glazing area distribution.

Sample code:

python

Copy code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import OneHotEncoder

# Load data

data = pd.read\_csv('2024\_energy\_efficiency\_data.csv')

# Shuffle and split data

train\_data, test\_data = train\_test\_split(data, test\_size=0.25, shuffle=True)

# One-hot encoding

encoder = OneHotEncoder(sparse=False)

categorical\_features = ['Orientation', 'Glazing area distribution']

encoded\_train = encoder.fit\_transform(train\_data[categorical\_features])

1. **Build the Neural Network**:
   * **Define the architecture**: You will need to manually implement the network without using high-level deep learning libraries (like TensorFlow or PyTorch).
     + Define layers (number of hidden layers and neurons) using numpy to initialize weights and biases.
     + Use activation functions like Sigmoid or Tanh.
   * **Forward Propagation**: Implement the logic for passing the input through the network layers.
   * **Backward Propagation**: Compute the gradients using the chain rule and update the weights.
   * **Loss Function**: Use Sum of Squares Error (SSE) and compute the Root Mean Squared Error (RMSE).

Key points to implement:

python

Copy code

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

# Initialize weights and biases

input\_size = 8 # Number of features

hidden\_size = 10

output\_size = 1

W1 = np.random.randn(input\_size, hidden\_size)

b1 = np.random.randn(hidden\_size)

W2 = np.random.randn(hidden\_size, output\_size)

b2 = np.random.randn(output\_size)

# Forward propagation

def forward(X):

z1 = np.dot(X, W1) + b1

a1 = sigmoid(z1)

z2 = np.dot(a1, W2) + b2

return z2

1. **Train the Model**:
   * Define a training loop where you:
     + Perform forward propagation to get predictions.
     + Calculate the error.
     + Backpropagate to compute gradients.
     + Update weights using the stochastic gradient descent algorithm.
2. **Evaluate the Model**:
   * After training, compute the **RMS error** for both training and testing sets.
3. **Report the Results**:
   * **Network architecture**: Number of hidden layers, neurons per layer.
   * **Learning curve**: Plot the training loss over epochs.
   * **RMS Error**: Report both training and test errors.
   * **Regression Results**: Plot predicted vs actual values for both the training and testing sets.

**Step 3: Work on the Classification Task**

1. **Load the Ionosphere Dataset** (not provided here, but you need to obtain it).
   * Similar to the regression task, shuffle the data and split it into 80% training and 20% testing.
   * The last column represents labels, with two classes: 'g' (good) and 'b' (bad).
2. **Build a Classification Network**:
   * The architecture will be similar to the regression network, but the **output layer will use SoftMax** for classification.
   * Use the **Cross-Entropy loss function** for this task.

Key points:

python

Copy code

def softmax(x):

exp\_x = np.exp(x - np.max(x))

return exp\_x / np.sum(exp\_x, axis=1, keepdims=True)

def cross\_entropy(y\_pred, y\_true):

n\_samples = y\_true.shape[0]

res = -np.sum(y\_true \* np.log(y\_pred + 1e-9)) / n\_samples

return res

1. **Train the Classification Model**:
   * Follow a similar process to the regression task, using forward propagation, backpropagation, and weight updates.
2. **Report the Results**:
   * **Network architecture**: Layers and neurons.
   * **Learning curve**: Plot training loss over epochs.
   * **Training and test error rates**.
   * **Latent feature distribution** for different numbers of nodes in the hidden layers.

**Step 4: Submission**

* Compress your source code and report (with results, plots, architecture details) into a zip file named hw1\_YourStudentID.zip.
* Submit the file before the deadline.

By following these steps, you will be able to implement the deep neural network models required for both the regression and classification tasks in your homework​(2024\_Deep\_Learning\_HW1)​(2024\_HW1\_guideline).