Aprendizagem 2023
Homework II – Group 090
(ist1103335, ist1103573)

1.

Consider the problem of learning a regression model from 4 bivariate observations:

$$\left\{\begin{pmatrix}0.7\\-0.3\end{pmatrix}),\begin{pmatrix}0.4\\0.5\end{pmatrix},\begin{pmatrix}-0.2\\0.8\end{pmatrix},\begin{pmatrix}-0.4\\0.3\end{pmatrix}\right\} \text{ with targets } (0.8, 0.6, 0.3, 0.3)\,.$$

(a) Given the radial basis function, $\phi_j(x) = exp\left(-\frac{\|x-\mathbf{c}_j\|^2}{2}\right)$, that transforms the original space onto a new space characterized by the similarity of the original observations to the following data points:

$$\left\{\mathbf{c}_1 = \begin{pmatrix}0\\0\end{pmatrix}, \mathbf{c}_2 = \begin{pmatrix}1\\-1\end{pmatrix}, \mathbf{c}_3 = \begin{pmatrix}-1\\1\end{pmatrix}\right\}$$

Learn the Ridge regression (ℓ_2 regularization) using the closed solution with $\lambda = 0.1$.

For each observation $\mathbf{x}$, we'll compute the radial basis function features using the given formula, $\phi_j(x)$, where $\mathbf{c}_j$ are the data points $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$, and the target vector $\mathbf{z}$:

$$X = \begin{bmatrix}1 & 0.74826 & 0.74826 & 0.10127\\1 & 0.81465 & 0.27117 & 0.33121\\1 & 0.71177 & 0.09633 & 0.71177\\1 & 0.8825 & 0.16122 & 0.65377\end{bmatrix}, \mathbf{z} = \begin{pmatrix}0.8\\0.6\\0.3\\0.3\end{pmatrix}, \lambda = 0.1$$

Using the targets and the regularization, the regression can be learnt using:

$$w = \left(X^T \cdot X + \lambda I\right)^{-1} \cdot X^T \cdot \mathbf{z}$$

Applying the above formula, we get:

$$w = \begin{pmatrix}0.33914\\0.19945\\0.40096\\-0.296\end{pmatrix}$$

(b) Compute the training RMSE for the learned regression. First, we calculate the predicted values:

$$\hat{z} = X \cdot w = \begin{pmatrix}0.75844\\0.51232\\0.30905\\0.38629\end{pmatrix}$$

Then, we aplly the RMSE formula:

$$RMSE(\hat{z}, z) = \sqrt{\frac{1}{n}\Sigma\left(z_i - \hat{z}_i\right)^2}, RMSE = \mathbf{0.06508}$$

2.

Consider a MLP classifier of three outcomes – A, B, and C – characterized by the weights:

$$W^{[1]} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$
$$W^{[2]} = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad b^{[2]} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$
$$W^{[3]} = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 1 \end{bmatrix}, \quad b^{[3]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The **activation function** is:

$$f(x) = \frac{e^{0.5x-2} - e^{-0.5x+2}}{e^{0.5x-2} + e^{-0.5x+2}} = \tanh(0.5x - 2)$$

for every unit, and **Squared Error loss**:

$$\frac{1}{2} \|\mathbf{z} - \hat{\mathbf{z}}\|_2^2$$

Perform one batch gradient descent update (with learning rate $\eta = 0.1$) for training observations:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

with targets B and A, respectively.

In our case, the target output for $\mathbf{x}_1$ is B:

$$B = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

And the target output for $\mathbf{x}_2$ is A:

$$A = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

The activation function has a well-known derivative that we will use later:

$$\text{activation\_derivative}(x) = \frac{\partial \tanh(0.5x - 2)}{\partial x} = 0.5(1 - \tanh^2(0.5x - 2))$$

First we do **Forward Propagation** for $X^{[0]}$ being $\mathbf{x}_1$ and then $\mathbf{x}_2$:

$$\begin{cases} Z^{[1]} = \mathbf{W}^{[1]}X^{[0]}{+}\mathbf{b}^{[1]} \\ X^{[1]} = tanh\left(0.5Z^{[1]}\text{-}2\right) \end{cases}, \begin{cases} Z^{[2]} = \mathbf{W}^{[2]}X^{[1]}{+}\mathbf{b}^{[2]} \\ X^{[2]} = tanh\left(0.5Z^{[2]}\text{-}2\right) \end{cases}, \begin{cases} Z^{[3]} = \mathbf{W}^{[3]}X^{[2]}{+}\mathbf{b}^{[3]} \\ X^{[3]} = tanh\left(0.5Z^{[3]}\text{-}2\right) \end{cases}$$

For example, with $X^{[0]} = \mathbf{x}_1$:

$$Z^{[1]}_{\mathbf{x}_1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix}$$

$$X^{[1]}_{\mathbf{x}_1} = tanh\left(0.5\begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix} \text{ - } 2\right) = \begin{bmatrix} 0.4621 \\ 0.7616 \\ 0.4621 \end{bmatrix}$$

With the forward propagation completed for the input $\mathbf{x}_1$ and $\mathbf{x}_2$, we have obtained two $\mathbf{X}^{[3]}$, which represents the predicted outputs of the network for these inputs.

The next step is to compute the gradient of the error function with respect to the network's weights:

$$\frac{\partial E(\mathbf{z}, \hat{\mathbf{z}})}{\partial W^{[l]}} = \frac{\partial E(\mathbf{z}, \hat{\mathbf{z}})}{\partial X^{[l]}} \times \frac{\partial X^{[l]}}{\partial W^{[l]}} = \delta^{[l]} \cdot (X^{[l-1]})^T$$

For which we need to calculate $\delta^{[L]}$ for the last layer:

$$\delta^{[L]} = \frac{\partial E(\mathbf{z}, \hat{\mathbf{z}})}{\partial \hat{\mathbf{z}}} \odot activation\_derivative(Z^{[l]})$$

Where $\frac{\partial E(\mathbf{z}, \hat{\mathbf{z}})}{\partial \hat{\mathbf{z}}}$ is given by:

$$\frac{\partial E(\mathbf{z}, \hat{\mathbf{z}})}{\partial \hat{\mathbf{z}}} = \frac{\partial(\frac{1}{2}\left\|\mathbf{z} - \hat{\mathbf{z}}\right\|_2^2)}{\partial \hat{\mathbf{z}}} = \hat{\mathbf{z}} - \mathbf{z} = \mathbf{X}^{[3]} - \mathbf{z}$$

Where $\mathbf{z}$ represents the target output (target A or B) and $\hat{\mathbf{z}}$ is the predicted output from the network ($X^{[3]}_{\mathbf{x}_1}$ and $X^{[3]}_{\mathbf{x}_2}$).

And the $\delta^{[l]}$ for the remaining layers:

$$\delta^{[l]} = \left(\mathbf{W}^{[l+1]}\right)^T \delta^{[l+1]} \odot \text{activation\_derivative}(\mathbf{Z}^{[2]})$$

Now, we'll calculate the $\delta^{[l]}$ for each layer using the formulas we found:

- For the output layer:

$$\delta^{[3]} = (\mathbf{X}^{[3]} - \mathbf{z}) \odot \text{activation\_derivative}\left(\mathbf{Z}^{[3]}\right)$$

$$\mathbf{x}_1 : \delta^{[3]}_{\mathbf{x}_1} = \begin{bmatrix} 0.0068 \\ -0.3177 \\ 0.0068 \end{bmatrix} \qquad \mathbf{x}_2 : \delta^{[3]}_{\mathbf{x}_2} = \begin{bmatrix} -2.6596 \times 10^{-2} \\ 3.3696 \times 10^{-6} \\ 1.8046 \times 10^{-4} \end{bmatrix}$$

- For the other layers:

$$\delta^{[2]} = \left(\mathbf{W}^{[3]}\right)^T \delta^{[3]} \odot \text{activation\_derivative}(\mathbf{Z}^{[2]})$$

$$\mathbf{x}_1 : \delta^{[2]}_{\mathbf{x}_1} = \begin{bmatrix} -0.3745 \\ -0.1016 \end{bmatrix} \qquad \mathbf{x}_2 : \delta^{[2]}_{\mathbf{x}_2} = \begin{bmatrix} -1.1509 \times 10^{-5} \\ -1.7290 \times 10^{-4} \end{bmatrix}$$

$$\delta^{[1]} = \left(\mathbf{W}^{[2]}\right)^T \delta^{[2]} \odot \text{activation\_derivative}(\mathbf{Z}^{[1]})$$

$$\mathbf{x}_1 : \delta^{[1]}_{\mathbf{x}_1} = \begin{bmatrix} -0.1872 \\ -0.3359 \\ -0.1872 \end{bmatrix} \qquad \mathbf{x}_2 : \delta^{[1]}_{\mathbf{x}_2} = \begin{bmatrix} -1.6662 \times 10^{-5} \\ -1.9782 \times 10^{-5} \\ -1.6662 \times 10^{-5} \end{bmatrix}$$

Now, we have the gradients of the Error function with respect to the weights:

$$\begin{aligned}
\sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{W}^{[3]}} &= \delta^{[3]}_{\mathbf{x}_1}(\mathbf{X}^{[2]}_{\mathbf{x}_1})^T + \delta^{[3]}_{\mathbf{x}_2}(\mathbf{X}^{[2]}_{\mathbf{x}_2})^T \\
&= \begin{bmatrix} 0.0068 \\ -0.3177 \\ 0.0068 \end{bmatrix} \begin{bmatrix} 0.4505 & -0.5764 \end{bmatrix} + \begin{bmatrix} -2.6596 \times 10^{-2} \\ 3.3696 \times 10^{-6} \\ 1.8046 \times 10^{-4} \end{bmatrix} \begin{bmatrix} -0.9996 & -0.9934 \end{bmatrix} \\
&= \begin{bmatrix} 0.0296 & 0.0225 \\ -0.1431 & 0.1831 \\ 0.0029 & -0.0041 \end{bmatrix}
\end{aligned}$$

$$\sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{W}^{[2]}} = \delta^{[2]}_{\mathbf{x}_1}(\mathbf{X}^{[1]}_{\mathbf{x}_1})^T + \delta^{[2]}_{\mathbf{x}_2}(\mathbf{X}^{[1]}_{\mathbf{x}_2})^T = \begin{bmatrix} -0.1730 & -0.2852 & -0.1730 \\ -0.0468 & -0.0771 & -0.0468 \end{bmatrix}$$

$$\sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{W}^{[1]}} = \delta^{[1]}_{\mathbf{x}_1}(\mathbf{X}^{[0]}_{\mathbf{x}_1})^T + \delta^{[1]}_{\mathbf{x}_2}(\mathbf{X}^{[0]}_{\mathbf{x}_2})^T = \begin{bmatrix} 0.0296 & 0.0225 \\ -0.1431 & 0.1831 \\ 0.0029 & -0.0041 \end{bmatrix}$$

We then calculate this gradients of the Error function with the input x_2 and target B, and added them up to get the one batch gradient.
We can now use these gradients to update the weights using one batch gradient descent. After updating the weights, the new weights will be:

$$\mathbf{W}^{[3]'} = \mathbf{W}^{[3]} - \eta \sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{W}^{[3]}}$$

$$\mathbf{W}^{[2]'} = \mathbf{W}^{[2]} - \eta \sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{W}^{[2]}}$$

$$\mathbf{W}^{[1]'} = \mathbf{W}^{[1]} - \eta \sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{W}^{[1]}}$$

Therefore the updated weights are:

$$\mathbf{W}^{[1]'} = \begin{bmatrix} 1.0187 & 1.0187 & 1.0187 & 1.0187 \\ 1.0336 & 1.0336 & 2.0336 & 1.0336 \\ 1.0187 & 1.0187 & 1.0187 & 1.0187 \end{bmatrix}$$

$$\mathbf{W}^{[2]'} = \begin{bmatrix} 1.0173 & 4.0285 & 1.0173 \\ 1.0047 & 1.0077 & 1.0047 \end{bmatrix}$$

$$\mathbf{W}^{[3]'} = \begin{bmatrix} 0.9970 & 0.9977 \\ 3.014 & 0.9817 \\ 0.9997 & 1.0004 \end{bmatrix}$$

And now, to update the biases, we need to calculate:

$$\frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[l]}} = \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{Z}^{[l]}} \times \frac{\partial Z^{[l]}}{\partial \mathbf{b}^{[l]}} = \delta^{[l]} \times 1 = \delta^{[l]}$$

With the previous formula, we calculate:

$$\sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[1]}} = \delta^{[1]}_{\mathbf{x}_1} + \delta^{[1]}_{\mathbf{x}_2} = \begin{bmatrix} -0.1872 \\ -0.3359 \\ -0.1872 \end{bmatrix}$$

$$\sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[2]}} = \delta^{[2]}_{\mathbf{x}_1} + \delta^{[2]}_{\mathbf{x}_2} = \begin{bmatrix} -0.3745 \\ -0.1017 \end{bmatrix}$$

$$\sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[3]}} = \delta^{[3]}_{\mathbf{x}_1} + \delta^{[3]}_{\mathbf{x}_2} = \begin{bmatrix} -0.0198 \\ -0.3177 \\ 0.0070 \end{bmatrix}$$

We can now use these gradients to update the biases using one batch gradient descent. After updating the biases, the new weights will be:

$$\mathbf{b}^{[1]'} = \mathbf{b}^{[1]} - \eta \sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[1]}}$$

$$\mathbf{b}^{[2]'} = \mathbf{b}^{[2]} - \eta \sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[2]}}$$

$$\mathbf{b}^{[3]'} = \mathbf{b}^{[3]} - \eta \sum_{k=1}^{2} \frac{\partial E\left(Z_k, \hat{Z}_k\right)}{\partial \mathbf{b}^{[3]}}$$

Therefore the updated biases are:

$$\mathbf{b}^{[1]'} = \begin{bmatrix} 1.0187 \\ 1.0336 \\ 1.0187 \end{bmatrix}$$

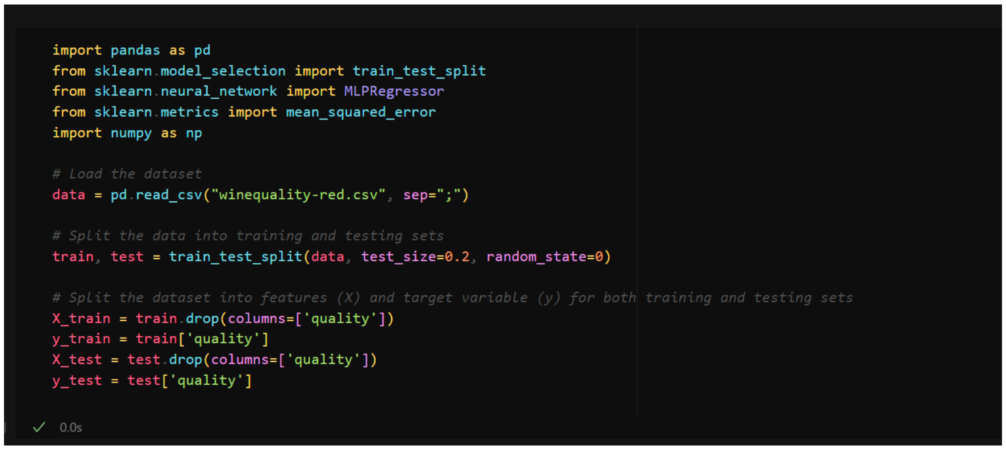
$$\mathbf{b}^{[2]'} = \begin{bmatrix} 1.0374 \\ 1.0102 \end{bmatrix}$$

$$\mathbf{b}^{[3]'} = \begin{bmatrix} 1.0020 \\ 1.0318 \\ 0.9993 \end{bmatrix}$$

II. Programming and critical analysis

Consider the winequality-red.csv dataset (available at the webpage) where the goal is to estimate the quality (sensory appreciation) of a wine based on physicochemical inputs. Using an 80-20 training-test split with a fixed seed (random_state = 0), you are asked to learn MLP regressors to answer the following questions.

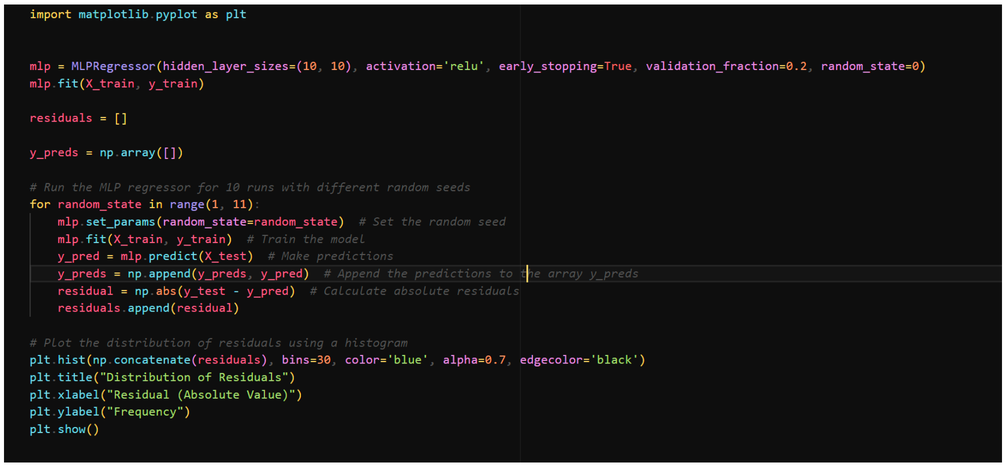
For the following questions we prepared the data like this:



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# Load the dataset
data = pd.read_csv("winequality-red.csv", sep=";")

# Split the data into training and testing sets
train, test = train_test_split(data, test_size=0.2, random_state=0)

# Split the dataset into features (X) and target variable (y) for both training and testing sets
X_train = train.drop(columns=['quality'])
y_train = train['quality']
X_test = test.drop(columns=['quality'])
y_test = test['quality']
```

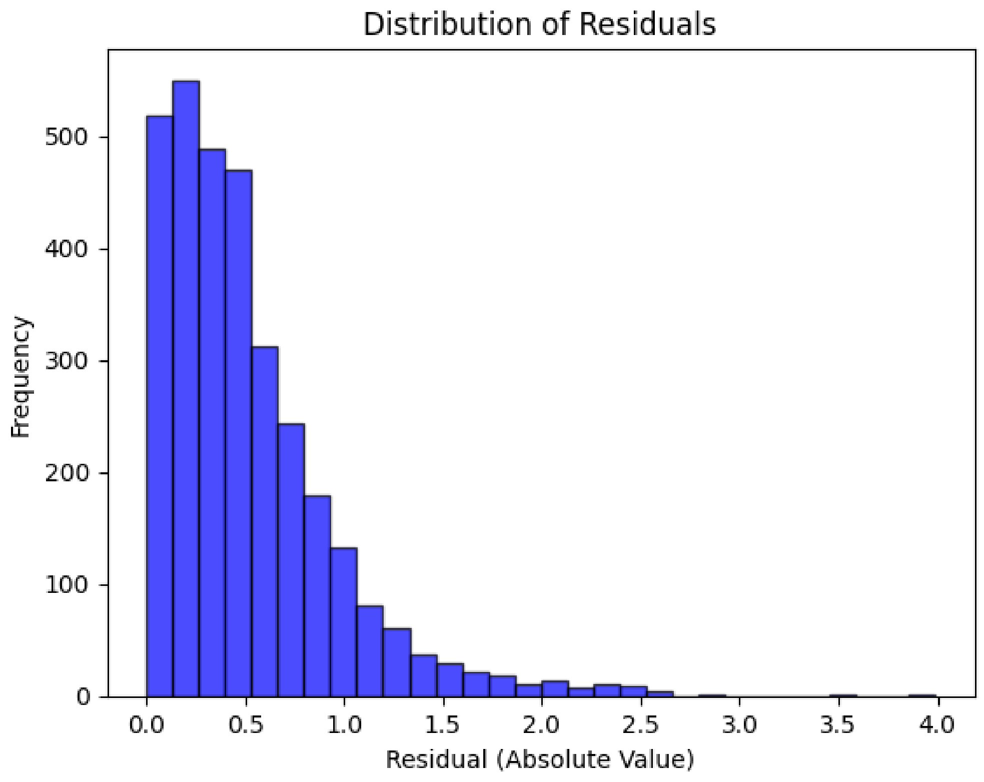
✓ 0.0s

(1) Learn a MLP regressor with 2 hidden layers of size 10, rectifier linear unit activation on all nodes, and early stopping with 20 % of training data set aside for validation. All remaining parameters (e.g., loss, batch size, regularization term, solver) should be set as default. Plot the distribution of the residues (in absolute value) using a histogram. To plot the histogram we used the following program:



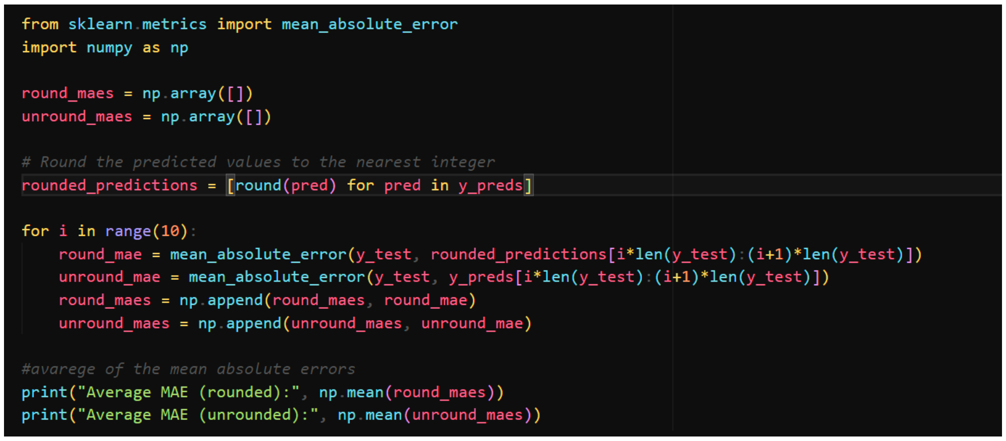
```python
import matplotlib.pyplot as plt


mlp = MLPRegressor(hidden_layer_sizes=(10, 10), activation='relu', early_stopping=True, validation_fraction=0.2, random_state=0)
mlp.fit(X_train, y_train)

residuals = []

y_preds = np.array([])

# Run the MLP regressor for 10 runs with different random seeds
for random_state in range(1, 11):
    mlp.set_params(random_state=random_state)  # Set the random seed
    mlp.fit(X_train, y_train)  # Train the model
    y_pred = mlp.predict(X_test)  # Make predictions
    y_preds = np.append(y_preds, y_pred)  # Append the predictions to the array y_preds
    residual = np.abs(y_test - y_pred)  # Calculate absolute residuals
    residuals.append(residual)

# Plot the distribution of residuals using a histogram
plt.hist(np.concatenate(residuals), bins=30, color='blue', alpha=0.7, edgecolor='black')
plt.title("Distribution of Residuals")
plt.xlabel("Residual (Absolute Value)")
plt.ylabel("Frequency")
plt.show()
```

We obtained the following histogram:



(2) Since we are in the presence of an integer regression task, a recommended trick is to round and bound estimates. Assess the impact of these operations on the MAE of the MLP learned in the previous question.

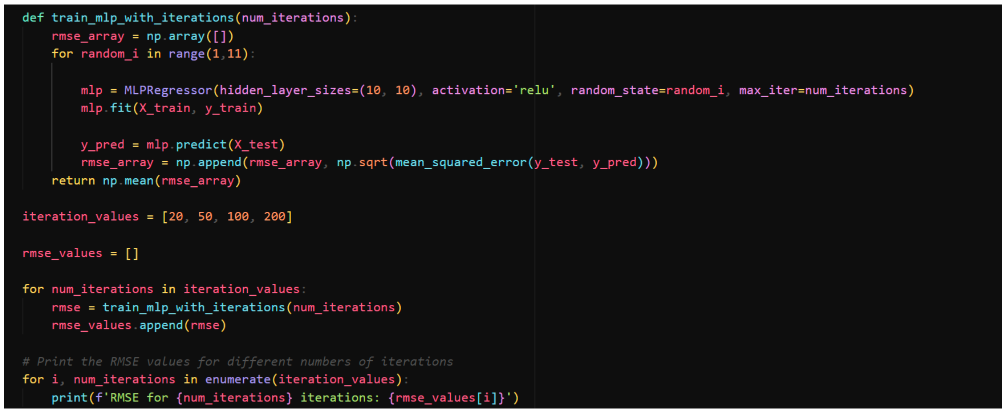
We performed the MAE with this:



```
from sklearn.metrics import mean_absolute_error
import numpy as np

round_maes = np.array([])
unround_maes = np.array([])

# Round the predicted values to the nearest integer
rounded_predictions = [round(pred) for pred in y_preds]

for i in range(10):
    round_mae = mean_absolute_error(y_test, rounded_predictions[i*len(y_test):(i+1)*len(y_test)])
    unround_mae = mean_absolute_error(y_test, y_preds[i*len(y_test):(i+1)*len(y_test)])
    round_maes = np.append(round_maes, round_mae)
    unround_maes = np.append(unround_maes, unround_mae)

#avarege of the mean absolute errors
print("Average MAE (rounded):", np.mean(round_maes))
print("Average MAE (unrounded):", np.mean(unround_maes))
```

The MAE for the rounded predictions is **0.4398** which is lower than the MAE for the unrounded predictions that is **0.5097**. This indicates that **rounding** the model's predictions to the nearest integer has **improved the model's performance** in terms of MAE.

(3) Similarly assess the impact on RMSE from replacing early stopping by a well-defined number of iterations in $\{20, 50, 100, 200\}$ (where one iteration corresponds to a batch).



```python
def train_mlp_with_iterations(num_iterations):
    rmse_array = np.array([])
    for random_i in range(1,11):

        mlp = MLPRegressor(hidden_layer_sizes=(10, 10), activation='relu', random_state=random_i, max_iter=num_iterations)
        mlp.fit(X_train, y_train)

        y_pred = mlp.predict(X_test)
        rmse_array = np.append(rmse_array, np.sqrt(mean_squared_error(y_test, y_pred)))
    return np.mean(rmse_array)

iteration_values = [20, 50, 100, 200]

rmse_values = []

for num_iterations in iteration_values:
    rmse = train_mlp_with_iterations(num_iterations)
    rmse_values.append(rmse)

# Print the RMSE values for different numbers of iterations
for i, num_iterations in enumerate(iteration_values):
    print(f'RMSE for {num_iterations} iterations: {rmse_values[i]}')
```

With this programm we calculated de following RMSE values:

RMSE for 20 iterations: 1.4039789509925442
RMSE for 50 iterations: 0.7996073631460568
RMSE for 100 iterations: 0.6940361469112143
RMSE for 200 iterations: 0.6554543932216474

(4) Critically comment on the results obtained in the previous question.

We can observe that as the number of training iterations increases, the RMSE decreases, which is a positive sign. Lower RMSE values suggest that **the model is improving** its ability to predict the target variable more accurately.