Aprendizagem 2023
Homework III – Group 28

Gonçalo Bárias (ist1103124) & Raquel Braunschweig (ist1102624)

Part I: Pen and Paper

For questions in this group, show your numerical results with 5 decimals or scientific notation.
Hint: we highly recommend the use of `numpy` (e.g., `linalg.pinv` for inverse) or other programmatic facilities to support the calculus involved in both questions (1) and (2).

Note: The intermediate values and matrices, in both exercises, show values rounded to 5 decimal places, but all intermediate calculations have been carried out without rounding. To help with the calculations of this Part, and as suggested in the statement, we used `numpy`. The code for both exercises can be found in the Jupyter notebook `G028.ipynb`.

1. **Consider the problem of learning a regression model from 4 bivariate observations**

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

   (a) **Given the radial basis function,** $\phi_j(x) = \exp\left(-\frac{\|\mathbf{x}-c_j\|^2}{2}\right)$ **that transforms the original space onto a new space characterized by the similarity of the original observations to the following data points** $\left\{c_1 = \begin{bmatrix}0 & 0\end{bmatrix}^T, c_2 = \begin{bmatrix}1 & -1\end{bmatrix}^T, c_3 = \begin{bmatrix}-1 & 1\end{bmatrix}^T\right\}$**.**
   Learn the Ridge regression (l_2 **regularization) using the closed solution with** $\lambda = 0.1$**.**

   Considering the data points along with the 4 bivariate observations, we can calculate the values of $\phi_1(x), \phi_2(x)$ and $\phi_3(x)$ for each observation, with $\phi_0(x)$ always being equal to 1.

x	$\phi_0(x)$	$\phi_1(x)$	$\phi_2(x)$	$\phi_3(x)$
$(0.7, -0.3)$	1	0.74826	0.74826	0.10127
$(0.4, 0.5)$	1	0.81465	0.27117	0.33121
$(-0.2, 0.8)$	1	0.71177	0.09633	0.71177
$(-0.4, 0.3)$	1	0.88250	0.16122	0.65377

Table 1: Value of $\phi_j(x)$ for each observation, $j = 0, \ldots, 3$

   Our goal is to perform a regression where the prediction is given by:

$$\hat{z}(x, w) = w_0\,\phi_0(x) + w_1\,\phi_1(x) + w_2\,\phi_2(x) + w_3\,\phi_3(x) \tag{1}$$

   Since we want to learn a Ridge regression model, we will need to minimize the following regularized least-squares estimator:

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(z_i - w^T \cdot x_i)^2 + \frac{\lambda}{2}\|w\|_2^2 \tag{2}$$

   To minimize (2), we set its gradient equal to 0 and obtain the closed-form solution with $\lambda = 0.1$ in equation (3). In the equation, we have that Φ is the matrix after applying $\phi_j(x)$ to the observations (Table 1) and z is the vector of targets for the observations, $z = \begin{bmatrix}0.8 & 0.6 & 0.3 & 0.3\end{bmatrix}^T$.

$$\nabla E(w) = 0 \;\Leftrightarrow\; w = \left(\Phi^T\Phi + \lambda I\right)^{-1} \cdot \Phi^T z \;\Leftrightarrow\; w = \left(\Phi^T\Phi + 0.1I\right)^{-1} \cdot \Phi^T z \tag{3}$$

$$\Phi = \begin{bmatrix} 1.00000 & 0.74826 & 0.74826 & 0.10127 \\ 1.00000 & 0.81465 & 0.27117 & 0.33121 \\ 1.00000 & 0.71177 & 0.09633 & 0.71177 \\ 1.00000 & 0.88250 & 0.16122 & 0.65377 \end{bmatrix} \qquad \Phi^T = \begin{bmatrix} 1.00000 & 1.00000 & 1.00000 & 1.00000 \\ 0.74826 & 0.81465 & 0.71177 & 0.88250 \\ 0.74826 & 0.27117 & 0.09633 & 0.16122 \\ 0.10127 & 0.33121 & 0.71177 & 0.65377 \end{bmatrix}$$

Therefore, we can then calculate the remaining values, until we get the vector w:

$$\Phi^T\Phi = \begin{bmatrix} 4.00000 & 3.15718 & 1.27698 & 1.79802 \\ 3.15718 & 2.50897 & 0.99165 & 1.42916 \\ 1.27698 & 0.99165 & 0.66870 & 0.33955 \\ 1.79802 & 1.42916 & 0.33955 & 1.05399 \end{bmatrix}$$

$$\Phi^T\Phi + 0.1\,I = \begin{bmatrix} 4.10000 & 3.15718 & 1.27698 & 1.79802 \\ 3.15718 & 2.60897 & 0.99165 & 1.42916 \\ 1.27698 & 0.99165 & 0.76870 & 0.33955 \\ 1.79802 & 1.42916 & 0.33955 & 1.15399 \end{bmatrix}$$

$$\left(\Phi^T\Phi + 0.1I\right)^{-1} = \begin{bmatrix} 4.54826 & -3.77682 & -1.86117 & -1.86155 \\ -3.77682 & 5.98285 & -0.88543 & -1.26432 \\ -1.86117 & -0.88543 & 4.33276 & 2.72156 \\ -1.86155 & -1.26432 & 2.72156 & 4.53204 \end{bmatrix}$$

$$\left(\Phi^T\Phi + 0.1I\right)^{-1} \cdot \Phi^T = \begin{bmatrix} 0.14105 & 0.35022 & 0.35575 & -0.30185 \\ -0.09064 & 0.43823 & -0.50361 & 0.53370 \\ 0.99394 & -0.50615 & -0.13690 & -0.16477 \\ -0.31222 & -0.65246 & 0.72647 & 0.42436 \end{bmatrix}$$

$$w = \left(\Phi^T\Phi + 0.1I\right)^{-1} \cdot \Phi^T z = \begin{bmatrix} 0.33914 & 0.19945 & 0.40096 & -0.29600 \end{bmatrix}^T$$

Having calculated the vector w, we now know the values of each weight in the regression:

$$w_0 = 0.33914 \qquad w_1 = 0.19945 \qquad w_2 = 0.40096 \qquad w_3 = -0.29600$$

Finally, replacing them in (1) gives us the regression expression:

$$\hat{z}(x) = 0.33914 + 0.19945\,\phi_1(x) + 0.40096\,\phi_2(x) - 0.29600\,\phi_3(x) \tag{4}$$

(b) **Compute the training RMSE for the learnt regression.**

The RMSE (Root Mean Square Error) is given by:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(z_i - \hat{z}_i)^2} \tag{5}$$

Here we have $N = 4$, because 4 is the number of samples in the dataset, z_i is the true label for the i-th sample and $\hat{z}_i$ is the predicted label for the i-th sample.

Using (4) from the previous item, we can determine $\hat{z}$ for the 4 observations:

$$\Phi = \begin{bmatrix} 1.00000 & 0.74826 & 0.74826 & 0.10127 \\ 1.00000 & 0.81465 & 0.27117 & 0.33121 \\ 1.00000 & 0.71177 & 0.09633 & 0.71177 \\ 1.00000 & 0.88250 & 0.16122 & 0.65377 \end{bmatrix} \wedge w = \begin{bmatrix} 0.33914 \\ 0.19945 \\ 0.40096 \\ -0.29600 \end{bmatrix} \Rightarrow \hat{z} = \Phi w = \begin{bmatrix} 0.75844 \\ 0.51232 \\ 0.30905 \\ 0.38629 \end{bmatrix}$$

Finally, we can take $z = \begin{bmatrix} 0.8 & 0.6 & 0.3 & 0.3 \end{bmatrix}^T$ along with $\hat{z}$ and calculate RMSE, using (5):

$$\text{RMSE} = \sqrt{\frac{1}{4} \times \sum_{i=1}^{4} (z_i - \hat{z}_i)^2} = \sqrt{\frac{1}{4} \times 0.01694} = \mathbf{0.06508}$$

2. **Consider a MLP classifier of three outcomes - A, B and C - characterized by the following weights and activation function for every unit:**

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

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

We also have the 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 obbervations $x_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$ and $x_2 = \begin{bmatrix} 1 & 0 & 0 & -1 \end{bmatrix}^T$ with targets B and A, respectively.

To begin, we initiate the process with the **forward propagation**. Below are the key equations to calculate the values of each layer, $x_i^{[p]}$, per observation:

$$z_i^{[p]} = W^{[p]} x_i^{[p-1]} + b^{[p]} \qquad\qquad x_i^{[p]} = f\left(z_i^{[p]}\right)$$

The function f is the activation function, for every layer, provided in the statement, $f\left(z_i^{[p]}\right) = \tanh\left(0.5\, z^{[p]} - 2\right)$. In this notation, the value p is the index of the MLP layer and n is the number of the observation, either 1 or 2.

We can now calculate the values of each node in the multi-layer perceptron with the initialized weights and biases provided in the statement, for each of the observations.

We will **start** with x_1:

$$z_1^{[1]} = W^{[1]} x_1^{[0]} + b^{[1]} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix}$$

$$x_1^{[1]} = f\left(z_1^{[1]}\right) = \tanh\left(0.5\, z_1^{[1]} - 2\right) \approx \begin{bmatrix} 0.46212 \\ 0.76159 \\ 0.46212 \end{bmatrix}$$

$$z_1^{[2]} = W^{[2]}\, x_1^{[1]} + b^{[2]} = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.46212 \\ 0.76159 \\ 0.46212 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 4.97061 \\ 2.68583 \end{bmatrix}$$

$$x_1^{[2]} = f\left(z_1^{[2]}\right) = \tanh\left(0.5\, z_1^{[2]} - 2\right) \approx \begin{bmatrix} 0.45048 \\ -0.57642 \end{bmatrix}$$

$$z_1^{[3]} = W^{[3]}\, x_1^{[2]} + b^{[3]} = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0.45048 \\ -0.57642 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 0.87406 \\ 1.77503 \\ 0.87406 \end{bmatrix}$$

$$x_1^{[3]} = f\left(z_1^{[3]}\right) = \tanh\left(0.5\, z_1^{[3]} - 2\right) \approx \begin{bmatrix} -0.9159 \\ -0.80494 \\ -0.9159 \end{bmatrix}$$

Next in line is x_2:

$$z_2^{[1]} = W^{[1]}\, x_2^{[0]} + b^{[1]} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x_2^{[1]} = f\left(z_2^{[1]}\right) = \tanh\left(0.5\, z_2^{[1]} - 2\right) \approx \begin{bmatrix} -0.90515 \\ -0.90515 \\ -0.90515 \end{bmatrix}$$

$$z_2^{[2]} = W^{[2]}\, x_2^{[1]} + b^{[2]} = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.90515 \\ -0.90515 \\ -0.90515 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} -4.43089 \\ -1.71544 \end{bmatrix}$$

$$x_2^{[2]} = f\left(z_2^{[2]}\right) = \tanh\left(0.5\, z_2^{[2]} - 2\right) \approx \begin{bmatrix} -0.99956 \\ -0.99343 \end{bmatrix}$$

$$z_2^{[3]} = W^{[3]}\, x_2^{[2]} + b^{[3]} = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} -0.99956 \\ -0.99343 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} -0.993 \\ -2.99212 \\ -0.993 \end{bmatrix}$$

$$x_2^{[3]} = f\left(z_2^{[3]}\right) = \tanh\left(0.5\, z_2^{[3]} - 2\right) \approx \begin{bmatrix} -0.98652 \\ -0.99816 \\ -0.98652 \end{bmatrix}$$

Now that we have obtained the values for our observations, considering the initial weights and biases, we can initiate the **back propagation** process to perform a batch gradient descent update (with learning rate $\eta = 0.1$).

The updated values for the weights and biases can be obtained by the following equations:

$$W_{new}^{[i]} = W_{old}^{[i]} + \Delta W^{[i]} = W_{old}^{[i]} - \eta \frac{\partial E}{\partial W^{[i]}} = W_{old}^{[i]} - 0.1 \frac{\partial E}{\partial W^{[i]}} \tag{6}$$

$$b_{new}^{[i]} = b_{old}^{[i]} + \Delta b^{[i]} = b_{old}^{[i]} - \eta \frac{\partial E}{\partial b^{[i]}} = b_{old}^{[i]} - 0.1 \frac{\partial E}{\partial b^{[i]}} \tag{7}$$

We first need to take into account the loss function, which, as per the statement, is the half squared error loss, defined as follows:

$$E(w) = \frac{1}{2}\|\mathbf{z} - \hat{\mathbf{z}}\|_2^2$$

The next step involves computing its derivatives by applying the **chain rule**:

$$\frac{\partial E}{\partial W^{[p]}} = \sum_{i=1}^{2} \left(\frac{\partial E}{\partial x_i^{[p]}} \circ \frac{\partial x_i^{[p]}}{\partial z_i^{[p]}} \cdot \frac{\partial z_i^{[p]}}{\partial W^{[p]}} \right) \qquad \frac{\partial E}{\partial b^{[p]}} = \sum_{i=1}^{2} \left(\frac{\partial E}{\partial x_i^{[p]}} \circ \frac{\partial x_i^{[p]}}{\partial z_i^{[p]}} \cdot \frac{\partial z_i^{[p]}}{\partial b^{[p]}} \right)$$

This can be broken down into:

$$\frac{\partial E}{\partial x_i^{[p]}} \circ \frac{\partial x_i^{[p]}}{\partial z_i^{[p]}} = \delta_i^{[p]}$$

$$\frac{\partial E}{\partial x_i^{[3]}} = \frac{1}{2} \times 2 \times \left(x_i^{[3]} - t_i \right) = x_i^{[3]} - t_i$$

$$\frac{\partial E}{\partial x_i^{[p]}} = \frac{\partial z_i^{[p+1]}}{\partial x_i^{[p]}} \cdot \underbrace{\frac{\partial x_i^{[p+1]}}{\partial z_i^{[p+1]}} \circ \frac{\partial E}{\partial x_i^{[p+1]}}}_{\delta_i^{[p+1]}} = \left(W^{[p+1]} \right)^T \cdot \delta_i^{[p+1]}, \quad p \neq 3 \quad \text{(using recursion)}$$

$$\begin{aligned} \frac{\partial x_i^{[p]}}{\partial z_i^{[p]}} &= f'\left(z_i^{[p]} \right) = \tanh'\left(0.5 \times z_i^{[p]} - 2 \right) \times \frac{d}{dz_i^{[p]}} \left(0.5 \times z_i^{[p]} - 2 \right) \\ &= \operatorname{sech}^2\left(0.5 \times z_i^{[p]} - 2 \right) \times 0.5 \end{aligned}$$

$$\frac{\partial z_i^{[p]}}{\partial W^{[p]}} = \left(x_i^{[p-1]} \right)^T \qquad \frac{\partial z_i^{[p]}}{\partial b^{[p]}} = 1$$

Therefore, the derivatives of the loss function are given by:

$$\frac{\partial E}{\partial W^{[p]}} = \sum_{i=1}^{2} \left(\delta_i^{[p]} \cdot \left(x_i^{[p-1]} \right)^T \right) \qquad \frac{\partial E}{\partial b^{[p]}} = \sum_{i=1}^{2} \left(\delta_i^{[p]} \right) \tag{8}$$

When computing δ, we split it into two separate cases: if it is in last layer (9) for $p = 3$, or in non-external layers (10) for any $p \neq 3$:

$$\delta_i^{[3]} = \left(x_i^{[3]} - t_i \right) \circ \operatorname{sech}^2\left(0.5 \times z_i^{[3]} - 2 \right) \times 0.5 \tag{9}$$

$$\delta_i^{[p]} = \left(\left(W^{[p+1]} \right)^T \cdot \delta_i^{[p+1]} \right) \circ \operatorname{sech}^2\left(0.5 \times z_i^{[p]} - 2 \right) \times 0.5, \quad p \neq 3 \tag{10}$$

Since, the `tanh` function has a codomain in $[-1, 1]$ and since the targets for x_1 and x_2 are B and A, respectively, we can conclude the following encodings for the outputs:

$$t_1 = \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}^T \qquad t_2 = \begin{bmatrix} 1 & -1 & -1 \end{bmatrix}^T$$

Now, we have all we need to compute the values for the deltas, δ, by replacing the equation on (9) for $p = 3$, and (10) for any $p \neq 3$:

$$\begin{aligned}
\delta_1^{[3]} &= \left(x_1^{[3]} - t_1\right) \circ \operatorname{sech}^2\left(0.5 \times z_1^{[3]} - 2\right) \times 0.5 = \\
&= \left(\begin{bmatrix} -0.9159 \\ -0.80494 \\ -0.9159 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}\right) \circ \operatorname{sech}^2\left(0.5 \times \begin{bmatrix} 0.87406 \\ 1.77503 \\ 0.87406 \end{bmatrix} \times 0.5\right) = \\
&= \begin{bmatrix} 0.00678 \\ -0.31773 \\ 0.00678 \end{bmatrix} \\
\delta_2^{[3]} &= \left(x_2^{[3]} - t_2\right) \circ \operatorname{sech}^2\left(0.5 \times z_2^{[3]} - 2\right) \times 0.5 = \\
&= \left(\begin{bmatrix} -0.98652 \\ -0.99816 \\ -0.98652 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}\right) \circ \operatorname{sech}^2\left(0.5 \times \begin{bmatrix} -0.993 \\ -2.99212 \\ -0.993 \end{bmatrix} \times 0.5\right) = \\
&= \begin{bmatrix} -0.0266 \\ 0 \\ 0.00018 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\delta_1^{[2]} &= \left(\left(W^{[3]}\right)^T \cdot \delta_1^{[3]}\right) \circ \operatorname{sech}^2\left(0.5 \times z_1^{[2]} - 2\right) \times 0.5 = \\
&= \left(\begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} 0.00678 \\ -0.31773 \\ 0.00678 \end{bmatrix}\right) \circ \operatorname{sech}^2\left(0.5 \times \begin{bmatrix} 4.97061 \\ 2.68583 \end{bmatrix} - 2\right) \times 0.5 = \\
&= \begin{bmatrix} -0.37448 \\ -0.10156 \end{bmatrix} \\
\delta_2^{[2]} &= \left(\left(W^{[3]}\right)^T \cdot \delta_2^{[3]}\right) \circ \operatorname{sech}^2\left(0.5 \times z_2^{[2]} - 2\right) \times 0.5 = \\
&= \left(\begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} -0.0266 \\ 0 \\ -0.00018 \end{bmatrix}\right) \circ \operatorname{sech}^2\left(0.5 \times \begin{bmatrix} 4.97061 \\ 2.68583 \end{bmatrix} - 2\right) \times 0.5 = \\
&= \begin{bmatrix} -1 \times 10^{-5} \\ -1.7 \times 10^{-4} \end{bmatrix}
\end{aligned}$$

$$\delta_1^{[1]} = \left(\left(W^{[2]}\right)^T \cdot \delta_1^{[2]}\right) \circ \operatorname{sech}^2\left(0.5 \times z_1^{[1]} - 2\right) \times 0.5 =$$

$$= \left(\begin{bmatrix} 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} -0.37448 \\ -0.10156 \end{bmatrix} \right) \circ \operatorname{sech}^2 \left(0.5 \times \begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix} - 2 \right) \times 0.5 =$$

$$= \begin{bmatrix} -0.18719 \\ -0.33587 \\ -0.18719 \end{bmatrix}$$

$$\delta_2^{[1]} = \left(\left(W^{[2]} \right)^T \cdot \delta_2^{[2]} \right) \circ \operatorname{sech}^2 \left(0.5 \times z_2^{[1]} - 2 \right) \times 0.5 =$$

$$= \left(\begin{bmatrix} 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} -1 \times 10^{-5} \\ -1.7 \times 10^{-4} \end{bmatrix} \right) \circ \operatorname{sech}^2 \left(0.5 \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - 2 \right) \times 0.5 =$$

$$= \begin{bmatrix} -2 \times 10^{-5} \\ -2 \times 10^{-5} \\ -2 \times 10^{-5} \end{bmatrix}$$

Next, we will compute the derivatives of the loss function, employing the first equation at (8):

$$\frac{\partial E}{\partial W^{[1]}} = \sum_{i=1}^{2} \left(\delta_i^{[1]} \cdot \left(x_i^{[0]} \right)^T \right) = \left(\delta_1^{[1]} \cdot \left(x_1^{[0]} \right)^T \right) + \left(\delta_2^{[1]} \cdot \left(x_2^{[0]} \right)^T \right) =$$

$$= \left(\begin{bmatrix} -0.18719 \\ -0.33587 \\ -0.18719 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}^T \right) + \left(\begin{bmatrix} -2 \times 10^{-5} \\ -2 \times 10^{-5} \\ -2 \times 10^{-5} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}^T \right) =$$

$$= \begin{bmatrix} -0.18721 & -0.18719 & -0.18719 & -0.18717 \\ -0.33589 & -0.33587 & -0.33587 & -0.33585 \\ -0.18721 & -0.18719 & -0.18719 & -0.18717 \end{bmatrix}$$

$$\frac{\partial E}{\partial W^{[2]}} = \sum_{i=1}^{2} \left(\delta_i^{[2]} \cdot \left(x_i^{[1]} \right)^T \right) = \left(\delta_1^{[2]} \cdot \left(x_1^{[1]} \right)^T \right) + \left(\delta_2^{[2]} \cdot \left(x_2^{[1]} \right)^T \right) =$$

$$= \left(\begin{bmatrix} -0.37448 \\ 0.10156 \end{bmatrix} \cdot \begin{bmatrix} 0.46212 \\ 0.76159 \\ 0.46212 \end{bmatrix}^T \right) + \left(\begin{bmatrix} -1 \times 10^{-5} \\ -1.7 \times 10^{-4} \end{bmatrix} \cdot \begin{bmatrix} -0.90515 \\ -0.90515 \\ -0.90515 \end{bmatrix}^T \right) =$$

$$= \begin{bmatrix} -0.17304 & -0.28519 & -0.17304 \\ -0.04678 & -0.07719 & -0.04678 \end{bmatrix}$$

$$\frac{\partial E}{\partial W^{[3]}} = \sum_{i=1}^{2} \left(\delta_i^{[3]} \cdot \left(x_i^{[2]} \right)^T \right) = \left(\delta_1^{[3]} \cdot \left(x_1^{[2]} \right)^T \right) + \left(\delta_2^{[3]} \cdot \left(x_2^{[2]} \right)^T \right) =$$

$$= \left(\begin{bmatrix} 0.00678 \\ -0.31773 \\ 0.00678 \end{bmatrix} \cdot \begin{bmatrix} 0.45048 \\ -0.57642 \end{bmatrix}^T \right) + \left(\begin{bmatrix} -0.0266 \\ 0 \\ 0.00018 \end{bmatrix} \cdot \begin{bmatrix} -0.99956 \\ -0.99343 \end{bmatrix}^T \right) =$$

$$= \begin{bmatrix} 0.02964 & 0.02252 \\ -0.14314 & 0.18315 \\ 0.00287 & -0.00408 \end{bmatrix}$$

Next, we will employ the second equation at (8):

$$\frac{\partial E}{\partial b^{[1]}} = \sum_{i=1}^{2}\left(\delta_i^{[1]}\right) = \delta_1^{[1]} + \delta_2^{[1]} = \begin{bmatrix}-0.18719\\-0.33587\\-0.18719\end{bmatrix} + \begin{bmatrix}-2\times10^{-5}\\-2\times10^{-5}\\-2\times10^{-5}\end{bmatrix} = \begin{bmatrix}-0.18721\\-0.33589\\-0.18721\end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[2]}} = \sum_{i=1}^{2}\left(\delta_i^{[2]}\right) = \delta_1^{[2]} + \delta_2^{[2]} = \begin{bmatrix}-0.37448\\0.10156\end{bmatrix} + \begin{bmatrix}-1\times10^{-5}\\-1.7\times10^{-4}\end{bmatrix} = \begin{bmatrix}-0.37449\\-0.10173\end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[3]}} = \sum_{i=1}^{2}\left(\delta_i^{[3]}\right) = \delta_1^{[3]} + \delta_2^{[3]} = \begin{bmatrix}0.00678\\-0.31773\\0.00678\end{bmatrix} + \begin{bmatrix}-0.0266\\0\\0.00018\end{bmatrix} = \begin{bmatrix}-0.01982\\-0.31773\\0.00696\end{bmatrix}$$

The final step is to calculate the updated weights and biases.

By substituting the values into (6), we obtain the following updated weights:

$$W_{new}^{[1]} = \begin{bmatrix}1&1&1&1\\1&1&2&1\\1&1&1&1\end{bmatrix} - 0.1\begin{bmatrix}-0.18721&-0.18719&-0.18719&-0.18717\\-0.33589&-0.33587&-0.33587&-0.33585\\-0.18721&-0.18719&-0.18719&-0.18717\end{bmatrix} =$$

$$= \begin{bmatrix}1.01872&1.01872&1.01872&1.01872\\1.03359&1.03359&2.03359&1.03359\\1.01872&1.01872&1.01872&1.01872\end{bmatrix}$$

$$W_{new}^{[2]} = \begin{bmatrix}1&4&1\\1&1&1\end{bmatrix} - 0.1\begin{bmatrix}-0.17304&-0.28519&-0.17304\\-0.04678&-0.07719&-0.04678\end{bmatrix} =$$

$$= \begin{bmatrix}1.0173&4.02852&1.0173\\1.00468&1.00772&1.00468\end{bmatrix}$$

$$W_{new}^{[3]} = \begin{bmatrix}1&1\\3&1\\1&1\end{bmatrix} - 0.1\begin{bmatrix}0.02964&0.02252\\-0.14314&0.18315\\0.00287&-0.00408\end{bmatrix} =$$

$$= \begin{bmatrix}0.99704&0.99775\\3.01431&0.98169\\0.99971&1.00041\end{bmatrix}$$

Finally, by replacing the values into (7), we get the following updated biases:

$$b_{new}^{[1]} = \begin{bmatrix}1\\1\\1\end{bmatrix} - 0.1\begin{bmatrix}-0.18721\\-0.33589\\-0.18721\end{bmatrix} = \begin{bmatrix}1.01872\\1.03359\\1.01872\end{bmatrix}$$

$$b_{new}^{[2]} = \begin{bmatrix}1\\1\end{bmatrix} - 0.1\begin{bmatrix}-0.37449\\-0.10173\end{bmatrix} = \begin{bmatrix}1.03745\\1.01017\end{bmatrix}$$

$$b_{new}^{[3]} = \begin{bmatrix}1\\1\\1\end{bmatrix} - 0.1\begin{bmatrix}-0.01982\\-0.31773\\0.00696\end{bmatrix} = \begin{bmatrix}1.00198\\1.03177\\0.9993\end{bmatrix}$$

Part II: Programming and critical analysis

Considering 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 a 80-20 training-test split with a fixed seed (`random_state=0`), you are asked to learn MLP regressors to answer the following questions.

Given their stochastic behavior, average the performance of each MLP from 10 runs (for reproducibility consider seeding the MLPs with `random_state` $\in \{1..10\}$).

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.**

```python
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor

# Step 1: Load and prepare the dataset
data = pd.read_csv("./data/winequality-red.csv", sep=";")
X, y = data.drop("quality", axis=1), data["quality"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.2, random_state=0)

residues = []
for rs in range(1, 11):
    # Step 2: Learn the MLP regressor
    mlp = MLPRegressor(hidden_layer_sizes=(10, 10), activation="relu",
                       early_stopping=True, validation_fraction=0.2,
                       random_state=rs)
    mlp.fit(X_train, y_train)

    # Step 3: Collect the residues
    y_pred = mlp.predict(X_test)
    residues.extend(np.abs(y_pred - y_test))

# Step 4: Plot the distribution of the absolute residues
plt.figure(figsize=(8, 6))
plt.hist(residues, bins=30, color = "#00bfc4", edgecolor="black")
plt.xlabel("Absolute Residues")
plt.ylabel("Count")
plt.show()
```

The residues were calculated as the absolute difference between the predicted value and the expected value, for each observation. Due to the stochastic behavior of MLP regressors, we took the results of the MLP from 10 runs.

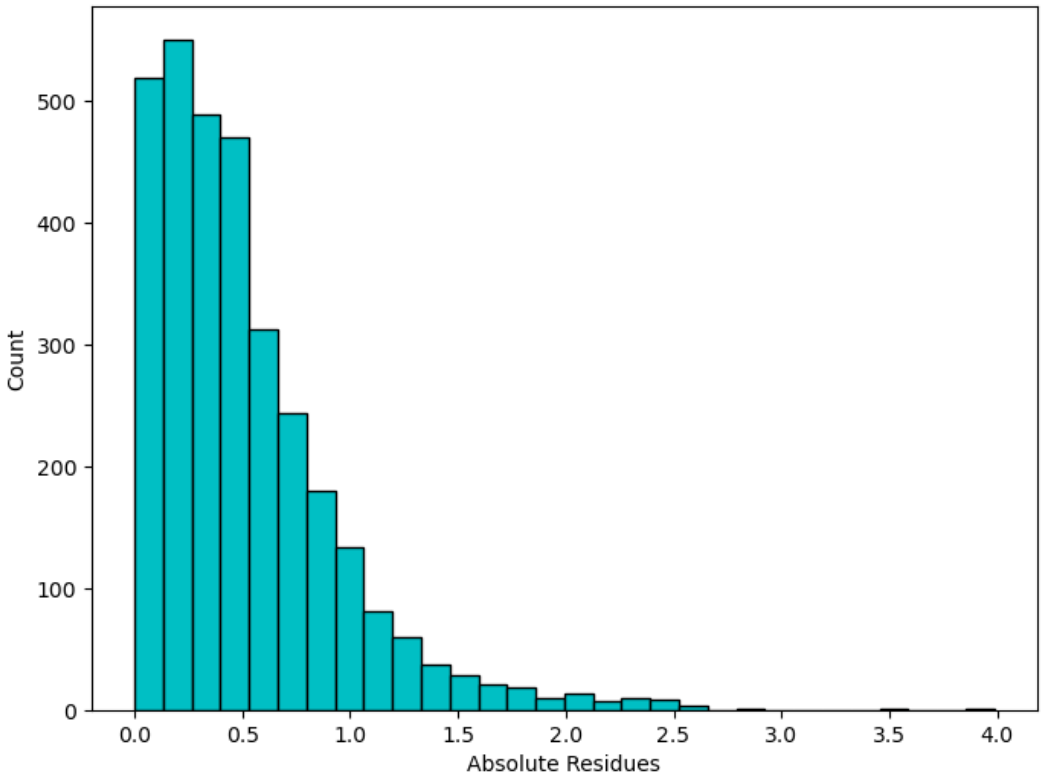

Figure 1: The distribution of absolute residues for MLP regression on wine quality prediction using an histogram with 30 bins

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

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

# Just like in the previous exercise
data = pd.read_csv("./data/winequality-red.csv", sep=";")
X, y = data.drop("quality", axis=1), data["quality"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.2, random_state=0)

maes, maes_rb = [], []
for rs in range(1, 11):
    # Train the MLP regressor
    mlp = MLPRegressor(hidden_layer_sizes=(10, 10), activation="relu",
                       early_stopping=True, validation_fraction=0.2,
                       random_state=rs)
    mlp.fit(X_train, y_train)

    # Apply rounding and bounding operations
    y_pred = mlp.predict(X_test)
    y_pred_rb = np.clip(np.round(y_pred), 1, 10)  # Bound between 1 and 10
```

```
    # Calculate MAE for both rounded and bounded predictions
    maes.append(mean_absolute_error(y_test, y_pred))
    maes_rb.append(mean_absolute_error(y_test, y_pred_rb))

# Print the MAE for both cases
print(f"MAE without operations: {np.mean(maes)}")
print(f"MAE with rounded and bounded predictions: {np.mean(maes_rb)}")
```

MAE without operations: 0.5097171955009515
MAE with rounded and bounded predictions: 0.43875

By rounding to the nearest unit and bounding the estimates between 1 and 10 (as per the *FAQ*), we get a lower Mean Absolute Error. The wine quality is an integer between 1 and 10, so it is expected that rounding and bounding the estimates gets them closer to the real integer values.

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).**

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

# Just like in the previous exercise
data = pd.read_csv("./data/winequality-red.csv", sep=";")
X, y = data.drop("quality", axis=1), data["quality"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.2, random_state=0)

n_iters, rmse_iters = [20, 50, 100, 200], []
for n_iter in n_iters:
    rmse_runs = []
    for rs in range(1, 11):
        # Train the MLP regressor with a specific number of iterations
        mlp = MLPRegressor(hidden_layer_sizes=(10, 10), activation="relu",
                           max_iter=n_iter, random_state=rs)
        mlp.fit(X_train, y_train)

        # Predict the target values on the test set and Calculate RMSE
        y_pred = mlp.predict(X_test)
        rmse_runs.append(mean_squared_error(y_test, y_pred, squared=False))
    rmse_iters.append(np.mean(rmse_runs))

# Print the RMSE for the different numbers of iterations
for i, n_iter in enumerate(n_iters):
    print(f"RMSE with {n_iter} iterations: {rmse_iters[i]}")
```

RMSE with 20 iterations: 1.4039789509925442
RMSE with 50 iterations: 0.7996073631460567
RMSE with 100 iterations: 0.6940361469112144
RMSE with 200 iterations: 0.6554543932216472

4. **Critically comment the results obtained in the previous question, hypothesizing at least one reason why early stopping favors and/or worsens performance.**

Before comparing the results, we need to calculate the RMSE for the early stopping case with a validation fraction of 20%:

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

# Just like in the previous exercise
data = pd.read_csv("./data/winequality-red.csv", sep=";")
X, y = data.drop("quality", axis=1), data["quality"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.2, random_state=0)

rmse = []
for rs in range(1, 11):
    # Train the MLP regressor
    mlp = MLPRegressor(hidden_layer_sizes=(10, 10), activation="relu",
                       early_stopping=True, validation_fraction=0.2,
                       random_state=rs)
    mlp.fit(X_train, y_train)

    # Calculate RMSE
    y_pred = mlp.predict(X_test)
    rmse.append(mean_squared_error(y_test, y_pred, squared=False))

# Print the RMSE with early stopping
print(f"RMSE with early stopping and validation_fraction = 0.2: {np.mean(rmse)}")
```

RMSE with early stopping and validation_fraction = 0.2: 0.6706527958221329

The model in exercises 1 and 2 uses early stopping with a validation fraction of 20%, which means that the model uses 80% of the data to train and the other 20% for validation. By reading the documentation about the sklearn.neural network.MLPRegressor, we can conclude that after each iteration, the model will calculate the validation score. If the score is not improving by at least `tol` (default value = 0.0001) for `n_iter_no_change` (default value = 10) iterations, the model will stop training.

In general, early stopping improves performance because it stops the MLP training before it overfits the training data. This happens, because if the validation score doesn't improve along a few iterations we start training unnecessary data, which would only optimize weights for the training data to the detriment of unseen data, and thus causing overfitting.

When we use a model without early stopping with a number of iterations equal to 20, 50 or 100, we can see that it doesn't have enough opportunity to learn the patterns in the data, which causes underfitting and so we get a larger error for this model in comparison to the early stopping one.

Conversely, we can also see that when we have 200 iterations, we get better performance than with early stopping ($0.671 > 0.655$), possibly because early stopping may be halting a little prematurely, which may be caused by the chosen validation sets granting validation scores that are not improving too early on. However, finding the right number of iterations can be difficult and so early stopping gives us a good estimate of when to stop.