Homework III – Group 019

Diogo Gaspar 99207, Rafael Oliveira 99311

Part I: Pen and paper

1. **Consider the basis function, $\phi_j(x) = x^j$, for performing a 3-order polynomial regression,**

$$\hat{z}(x, w) = \sum_{j=0}^{3} w_j \phi_j(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3.$$

   Learn the Ridge regression (l_2 regularization) on the transformed data space using the closed-form solution with $\lambda = 2$.

   We have in hands a **supervised learning** problem, with a given training dataset as shown below:

	y_1	z
x_1	0.8	24
x_2	1	20
x_3	1.2	10
x_4	1.4	13
x_5	1.6	12

   Table 1: Training dataset: y_1 as the input's (only) variable, z as the target variable

   We can note that in the statement's estimation function, $\hat{z}(x, w)$, x is a single-element vector (with its only entry being each sample's y_1 value). Therefore, it makes sense to "expand" the table above as follows, in order to have a broader representation of the values we'll end up using in the estimation function:

	y_1	y_1^2	y_1^3	z
x_1	0.8	0.64	0.512	24
x_2	1	1	1	20
x_3	1.2	1.44	1.728	10
x_4	1.4	1.96	2.744	13
x_5	1.6	2.56	4.096	12

   Table 2: Training dataset with additional information

   The equation below shows the closed-form solution for the Ridge regression problem, with $\lambda = 2$:

   $$w = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T z = (\Phi^T \Phi + 2I)^{-1} \Phi^T z$$

Here, Φ is the result of applying the basis function to our training dataset's inputs, such that:

$$\Phi = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_5) & \phi_2(x_5) & \phi_3(x_5) \end{bmatrix} = \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix}$$

We are now able to learn the given polynomial regression model, with $\lambda = 2$:

$$(\Phi^T\Phi + \lambda I)^{-1} = \left(\begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix}^T \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix} + \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} 0.34168753 & -0.1214259 & -0.07490231 & -0.00932537 \\ -0.1214259 & 0.3892078 & -0.09667718 & -0.07445624 \\ -0.07490231 & -0.09667718 & 0.37257788 & -0.17135047 \\ -0.00932537 & -0.07445624 & -0.17135047 & 0.17998796 \end{bmatrix}$$

$$\Phi^T z = \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix}^T \begin{bmatrix} 24 \\ 20 \\ 10 \\ 13 \\ 12 \end{bmatrix} = \begin{bmatrix} 79 \\ 88.6 \\ 105.96 \\ 134.392 \end{bmatrix}$$

$$w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T z = \begin{bmatrix} 7.0450759 \\ 4.64092765 \\ 1.96734046 \\ -1.30088142 \end{bmatrix}$$

Having learned the regression model, we can now use it to predict labels z for new samples!

2. **Compute the training RMSE for the learnt regression model.**

We know that the Root Mean Squared Error (RMSE) for a given regression model is defined as

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

where N 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. As stated in the previous question's statement, $\hat{z}$ is given by the matrix product $\Phi \cdot w$. We can, then, compute the RMSE for the training dataset as follows:

$$\hat{z} = \Phi \cdot w = \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix} \begin{bmatrix} 7.0450759 \\ 4.64092765 \\ 1.96734046 \\ -1.30088142 \end{bmatrix} = \begin{bmatrix} 11.35086463 \\ 12.35246259 \\ 13.19923625 \\ 13.8287433 \\ 14.17854143 \end{bmatrix}$$

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

$$= \sqrt{\frac{1}{5} \left((24 - 11.35086463)^2 + \ldots + (12 - 14.17854143)^2 \right)}$$

$$= 6.84329$$

3. **Consider a multi-layer perceptron characterized by one hidden layer with 2 nodes. Using the activation function $f(x) = e^{0.1x}$ on all units, all weights initialized as 1 (including biases), and the half squared error loss, perform one batch gradient descent update (with learning rate $\eta = 0.1$) for the first three observations (0.8), (1) and (1.2).**

Our multi-layer perceptron, considering the parameters stated above, should only have one output-node, since we're considering a regression problem aiming to predict a single output variable. As such, we can represent the network as follows:

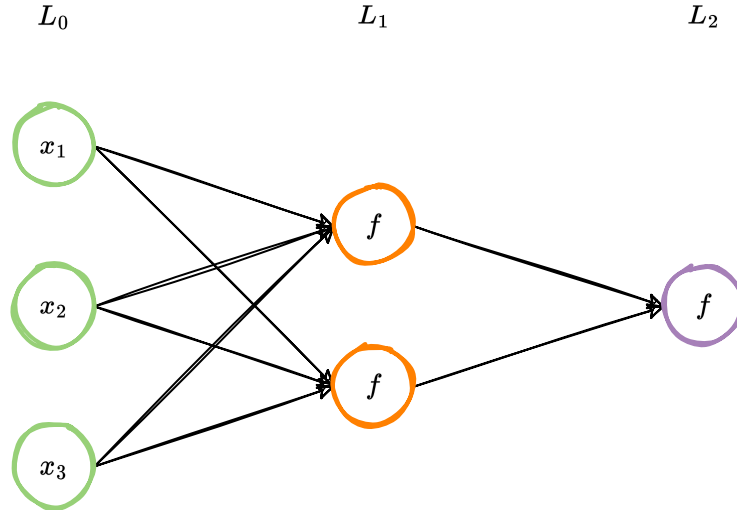L_0 $\qquad\qquad\qquad L_1$ $\qquad\qquad\qquad L_2$

Figure 1: Multi-layer perceptron shape

Each node in the hidden layer has an activation function $f(x) = e^{0.1x}$. Moreover, we know that the learning rule for the weights of the hidden layer is given by $\Delta w = -\eta \frac{\partial E}{\partial w}$ (with an analogous logic associated to biases), with $\eta = 0.1$ and E being the half squared error loss.

$$E = \frac{1}{2 * 2} \sum_{i=1}^{N} (z_i - \hat{z}_i)^2$$

A gradient descent update will require us to go through 3 phases: forward propagation, back propagation and updates (via gradient updates, updating biases and weights).

Starting by the forward propagation, and considering l as a given layer, we have that:

$$z^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)}, \quad x^{(l)} = f(z^{(l)})$$

We also know, from the question's statement, both the input nodes' values and weight/bias matrices:

$$x^{(0)} = \begin{bmatrix} 0.8 \\ 1 \\ 1.2 \end{bmatrix}, \quad w^{(1)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad w^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad b^{(2)} = \begin{bmatrix} 1 \end{bmatrix}$$

Applying the afore-mentioned equations in succession will lead us to the following results:

$$z^{(1)} = \begin{bmatrix} 4.00000000 \\ 4.00000000 \end{bmatrix}, \quad x^{(1)} = \begin{bmatrix} 1.4918247 \\ 1.4918247 \end{bmatrix}$$
$$z^{(2)} = \begin{bmatrix} 3.9836494 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} 1.48938747 \end{bmatrix}$$

We're now ready for the back propagation phase. Applying batch gradient descent, and looking at the half squared error loss, we have that:

$$\Delta w^{[l]} = -\eta \frac{\partial E}{\partial w^{[l]}}, \quad \Delta b^{[l]} = -\eta \frac{\partial E}{\partial b^{[l]}}$$

We'll want to propagate information backwards; for that, we'll need to use the chain rule, multiplying successive derivatives as we go backwards. With L being the **last layer** - will be useful for reusing previously calculated δ's - we have the following equalities:

$$\frac{\partial E}{\partial w^{[L]}} = \frac{\partial E}{\partial x^{[L]}} \circ \frac{\partial x^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]^T}}{\partial w^{[L]}}$$
$$\frac{\partial E}{\partial b^{[L]}} = \frac{\partial E}{\partial x^{[L]}} \circ \frac{\partial x^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]^T}}{\partial b^{[L]}}$$
$$\delta^{[L]} = \frac{\partial E}{\partial x^{[L]}} \circ \frac{\partial x^{[L]}}{\partial z^{[L]}}$$
$$\delta^{[l]} = \frac{\partial z^{[l+1]^T}}{\partial x^{[l]}} \cdot \delta^{[l+1]} \circ \frac{\partial x^{[l]}}{\partial z^{[l]}}$$

We'll now be able to start propagating backwards (considering the following equalities, derived both in class and in the curricular unit's book):

$$\frac{\partial E}{\partial x^{[2]}} = \frac{\partial \frac{1}{4} \sum_{i=1}^{N} (z_i - x_i^{[2]})^2}{\partial x^{[2]}} = \frac{x^{[2]} - z_i}{2}, \quad \frac{\partial x^{[l]}}{\partial z^{[l]}} = \frac{\partial e^{0.1z^{[l]}}}{\partial z^{[l]}} = 0.1 e^{0.1z^{[l]}}$$

$$\frac{\partial z^{[l]}}{\partial x^{[l-1]}} = w^{[l]}, \quad \frac{\partial z^{[l]}}{\partial b^{[l]}} = 1, \quad \frac{\partial z^{[l]}}{\partial w^{[l]}} = x^{[l-1]}$$

$$\delta^{[2]} = \frac{\partial E}{\partial x^{[2]}} \circ \frac{\partial x^{[2]}}{\partial z^{[2]}}$$
$$= \frac{(x^{[2]} - z_i)}{2} \circ 0.1 e^{0.1z^{[2]}}$$
$$= [0.244693735] \circ [0.1e^{0.39836494}]$$
$$= [0.03644437824]$$

$$\delta^{[1]} = \frac{\partial z^{[2]^T}}{\partial x^{[1]}} \cdot \delta^{[2]} \circ \frac{\partial x^{[1]}}{\partial z^{[1]}}$$
$$= \begin{bmatrix} 0.03644437824 \\ 0.03644437824 \end{bmatrix} \circ \begin{bmatrix} 0.1e^{0.4} \\ 0.1e^{0.4} \end{bmatrix}$$
$$= \begin{bmatrix} 0.005436862355 \\ 0.005436862355 \end{bmatrix}$$

In the last phase, we'll be updating our model: after computing the gradients, we'll be able to update weights and biases!

Starting with weight matrices:

$$\frac{\partial E}{\partial w^{[1]}} = \delta^{[1]} \cdot \frac{\partial z^{[1]^T}}{\partial w^{[1]}} = \delta^{[1]} \cdot x^{[0]^T} = \begin{bmatrix} 0.00434949 & 0.00543686 & 0.00652423 \\ 0.00434949 & 0.00543686 & 0.00652423 \end{bmatrix}$$

$$\frac{\partial E}{\partial w^{[2]}} = \delta^{[2]} \cdot \frac{\partial z^{[2]^T}}{\partial w^{[2]}} = \delta^{[2]} \cdot x^{[1]^T} = \begin{bmatrix} 0.05436862 & 0.05436862 \end{bmatrix}$$

$$w^{[1]} = w^{[1]} - \eta \cdot \frac{\partial E}{\partial w^{[1]}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} 0.00434949 & 0.00543686 & 0.00652423 \\ 0.00434949 & 0.00543686 & 0.00652423 \end{bmatrix}$$
$$= \begin{bmatrix} 0.999565051 & 0.999456314 & 0.999347577 \\ 0.999565051 & 0.999456314 & 0.999347577 \\ 0.999565051 & 0.999456314 & 0.999347577 \end{bmatrix}$$

$$w^{[2]} = w^{[2]} - \eta \cdot \frac{\partial E}{\partial w^{[2]}} = \begin{bmatrix} 1 & 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} 0.05436862 & 0.05436862 \end{bmatrix}$$
$$= \begin{bmatrix} 0.994563138 & 0.994563138 \end{bmatrix}$$

After updating weights, we'll update biases:

$$\frac{\partial E}{\partial b^{[1]}} = \delta^{[1]} \cdot \frac{\partial z^{[1]^T}}{\partial b^{[1]}} = \delta^{[1]} \cdot 1 = \delta^{[1]} = \begin{bmatrix} 0.00543686 \\ 0.00543686 \end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[2]}} = \delta^{[2]} \cdot \frac{\partial z^{[2]^T}}{\partial b^{[2]}} = \delta^{[2]} \cdot 1 = \delta^{[2]} = \begin{bmatrix} 0.03644438 \end{bmatrix}$$

$$b^{[1]} = b^{[1]} - \eta \cdot \frac{\partial E}{\partial b^{[1]}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} 0.00543686 \\ 0.00543686 \end{bmatrix}$$
$$= \begin{bmatrix} 0.99945631 \\ 0.99945631 \end{bmatrix}$$

$$b^{[2]} = b^{[2]} - \eta \cdot \frac{\partial E}{\partial b^{[2]}} = \begin{bmatrix} 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} 0.03644438 \end{bmatrix}$$
$$= \begin{bmatrix} 0.99635556 \end{bmatrix}$$

Part II: Programming and critical analysis

The code utilized to answer the following questions is available in this report's appendix.

4. **Compute the MAE of the three regressors: linear regression, MLP_1 and MLP_2.**

   We opted to utilize `sklearn`'s `mean_absolute_error` function to compute the MAE of the three regressors. The regressors were created as shown in the appendix (using `Ridge` and `MLPRegressor` with the respective parameters).

   We gathered the following results:

Regressor	MAE
Linear Regression (Ridge)	0.16283
MLP_1	0.06804
MLP_2	0.09781

   Table 3: Gathered Mean Absolute Errors for each specified regressor

5. **Plot the residues (in absolute value) using two visualizations: boxplots and histograms.**

   Each regressor's residues, calculated as the absolute difference between the predicted and actual values, were plotted using both boxplots and histograms (using, respectively, `seaborn`'s `boxplot` and `histplot` functions), as shown in this report's appendix (figure after the code).

6. **How many iterations were required for MLP_1 and MLP_2 to converge?**

   By definition, we know that MLPs converge when their parameters are no longer updated between epochs. With early stopping, however, we may actually stop training before the network has converged: using `sklearn`'s `MLPRegressor` implementation, it stops when the validation score has not improved in a given number of epochs by at least a given tolerance. **As such, given that MLP_1 utilizes early stopping, we're not able to know with certainty whether MLP_1 actually converged or not.** We can, however, assert that it effectively stopped training after 452 epochs.

   Regarding MLP_2, we can safely say that it converged after 77 epochs: it does not utilize early stopping, and as such should only stop after either a maximum number of epochs were reached (500 here, which isn't the case), or upon converging.

   As a side note, both iteration amounts were gathered by using the `print_regressor` method present in this code's appendix.

7. **What can be motivating the unexpected differences on the number of iterations? Hypothetize one reason underlying the observed performance differences between the MLPs.**

As has been noted in the previous question's answer, MLP_1 takes many more iterations to converge than MLP_2 - almost six times as many. It's probably worth emphasizing that the number of iterations in a batch gradient descent algorithm matches the number of epochs ran - the amount of times the algorithm goes through the entire dataset. Early stopping, however, is a technique which allows us to **stop an epoch** at a given point, not the entire algorithm.

Regarding performance, we were able to assert in the first question's answer that MLP_1 performed better than MLP_2 - it had a lower MAE, which effectively tells us that it was able to better approximate the function we were trying to learn to the data. With MLP_1 utilizing early stopping, it's likely that the regressor was able to avoid overfitting the data a bit better than MLP_2 - stopping at the "right point" of an epoch, instead of running the algorithm for all samples, could allow it to generalize better to unseen data.

Appendix

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io.arff import loadarff
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.neural_network import MLPRegressor
sns.set_style('darkgrid')

# Load data
data = loadarff('data/kin8nm.arff')
df = pd.DataFrame(data[0])

X = df.drop('y', axis=1).values
y = df['y'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=0
)

def predict(regressor):
    regressor.fit(X_train, y_train)
    return regressor.predict(X_test)

def plot_regressors_residues():
    """Utilized for answering question 2."""
    predictions = []
    descriptions = []
    for description, regressor in regressors.items():
        predictions.append(predict(regressor))
        descriptions.append(description)

    fig, (ax1, ax2) = plt.subplots(2, figsize=(8, 12))

    sns.histplot(
        data=pd.DataFrame({
            description: np.abs(y_test - prediction)
            for description, prediction in zip(descriptions, predictions)
        }),
        bins=30,
        ax=ax1,
        multiple='dodge',
    )
    ax1.set_title('Residues histogram for each regressor')
    ax1.set_xlabel('Residue')
    ax1.set_ylabel('Count')
    ax1.legend(descriptions)

    sns.boxplot(
        data=pd.DataFrame({
```

```python
55        description: np.abs(y_test - prediction)
56        for description, prediction in zip(descriptions, predictions)
57      }),
58      ax=ax2,
59      orient='h',
60    )
61    ax2.set_title('Residues boxplot for each regressor')
62    ax2.set_xlabel('Residue')
63    ax2.set_ylabel('Regressor')
64    ax2.legend(descriptions)
65    plt.savefig('assets/residues.png')
66
67
68  def print_regressor(regressor, description, y_pred):
69    """Utilized for answering questions 1. and 3."""
70    print(description)
71    print('MAE: {:.5f}'.format(mean_absolute_error(y_test, y_pred)))
72    if "MLP" in description:
73      print('Iterations: {}'.format(regressor.n_iter_))
74
75  regressors = {
76    "Ridge Regression": Ridge(alpha=0.1),
77    "MLP 1": MLPRegressor(
78      hidden_layer_sizes=(10, 10),
79      activation='tanh',
80      max_iter=500,
81      random_state=0,
82      early_stopping=True
83    ),
84    "MLP 2": MLPRegressor(
85      hidden_layer_sizes=(10, 10),
86      activation='tanh',
87      max_iter=500,
88      random_state=0
89    )
90  }
91
92  for description, regressor in regressors.items():
93    y_pred = predict(regressor)
94    print("---")
95    print_regressor(regressor, description, y_pred)
96
97  plot_regressors_residues()
```

Residues histogram for each regressor

Residues boxplot for each regressor