

Aprendizagem 2022  
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,  $\Phi$  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:

$$\begin{aligned} \hat{z} = \Phi \cdot w &= \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix} \begin{bmatrix} 7.0450759 \\ 4.64092765 \\ 1.96734046 \\ -1.30088142 \end{bmatrix} = \begin{bmatrix} 11.35086463 \\ 12.35246259 \\ 13.19923625 \\ 13.8287433 \\ 14.17854143 \end{bmatrix} \\ \text{RMSE} &= \sqrt{\frac{1}{5} \sum_{i=1}^5 (z_i - \hat{z}_i)^2} \\ &= \sqrt{\frac{1}{5} ((24 - 11.35086463)^2 + \dots + (12 - 14.17854143)^2)} \\ &= 6.84329 \end{aligned}$$

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.

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.

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

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 (where  $i$  matches the  $i$ -th sample):

$$z_i^{[l]} = w^{[l]} x_i^{[l]} + b^{[l]}, \quad x_i^{[l]} = f(z_i^{[l]})$$

We also know, from the question's statement, both the input nodes' values and weight/bias matrices (considering a column per sample for  $x$ ):

$$x^{[0]} = \begin{bmatrix} 0.8 & 1 & 1.2 \end{bmatrix}$$

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

Applying the afore-mentioned equations in succession will lead us to the following results (note that, just like with  $x$ , we'll also have one column per sample for  $z^{[l]}$ ):

$$\begin{aligned} z^{[1]} &= \begin{bmatrix} 1.8 & 2. & 2.2 \\ 1.8 & 2. & 2.2 \end{bmatrix} \\ z^{[2]} &= [3.39443473 \quad 3.44280552 \quad 3.49215346] \\ x^{[1]} &= \begin{bmatrix} 1.19721736 & 1.22140276 & 1.24607673 \\ 1.19721736 & 1.22140276 & 1.24607673 \end{bmatrix} \\ x^{[2]} &= [1.40416592 \quad 1.41097443 \quad 1.41795451] \end{aligned}$$

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  $\delta$ 's - we have the following equalities:

$$\begin{aligned} \delta_i^{[L]} &= \frac{\partial E}{\partial x_i^{[L]}} \circ \frac{\partial x_i^{[L]}}{\partial z_i^{[L]}} \\ \delta_i^{[l]} &= \frac{\partial z_i^{[l+1]T}}{\partial x_i^{[l]}} \cdot \delta_i^{[l+1]} \circ \frac{\partial x_i^{[l]}}{\partial z_i^{[l]}} \\ \frac{\partial E}{\partial w^{[L]}} &= \sum_{i=1}^N \delta_i^{[L]} \frac{\partial z^{[L]T}}{\partial w^{[L]}} \\ \frac{\partial E}{\partial b^{[L]}} &= \sum_{i=1}^N \delta_i^{[L]} \frac{\partial z_i^{[L]T}}{\partial b^{[L]}} \end{aligned}$$

Note that we'll be able to create a matrix  $\delta^{[l]}$  for each layer  $l$ , where each column  $i$  has the calculated  $\delta_i^{[l]}$ .

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

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

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

$$\begin{aligned} \delta^{[2]} &= \left[ \sum_{i=1}^N (x_i^{[2]} - z_i) \right] \circ 0.1 e^{0.1 z_i^{[2]}} \\ &= [-22.5958 \quad -18.5890 \quad -8.5820] \circ [0.14041659 \quad 0.14109744 \quad 0.14179545] \\ &= [-3.17283001 \quad -2.62286398 \quad -1.21689501] \end{aligned}$$

$$\begin{aligned} \delta^{[1]} &= \frac{\partial z^{[2]T}}{\partial x^{[1]}} \cdot \delta^{[2]} \circ \frac{\partial x^{[1]}}{\partial z^{[1]}} \\ &= \begin{bmatrix} -3.1728 & -2.6229 & -1.2169 \\ -3.1728 & -2.6229 & -1.2169 \end{bmatrix} \circ \begin{bmatrix} 0.11972174 & 0.12214028 & 0.12460767 \\ 0.11972174 & 0.12214028 & 0.12460767 \end{bmatrix} \\ &= \begin{bmatrix} -0.37985672 & -0.32035733 & -0.15163446 \\ -0.37985672 & -0.32035733 & -0.15163446 \end{bmatrix} \end{aligned}$$

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:

$$\frac{\partial E}{\partial w^{[1]}} = \sum_{i=1}^N \delta_i^{[1]} \cdot \frac{\partial z_i^{[1]T}}{\partial w^{[1]}} = \sum_{i=1}^N \delta_i^{[1]} \cdot x_i^{[0]T} = \begin{bmatrix} -0.80620405 \\ -0.80620405 \end{bmatrix}$$

$$\frac{\partial E}{\partial w^{[2]}} = \sum_{i=1}^N \delta_i^{[2]} \cdot x_i^{[1]T} = [-8.51848503 \quad -8.51848503]$$

$$w^{[1]} = w^{[1]} - \eta \cdot \frac{\partial E}{\partial w^{[1]}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} -0.80620405 \\ -0.80620405 \end{bmatrix} = \begin{bmatrix} 1.08062041 \\ 1.08062041 \end{bmatrix}$$

$$\begin{aligned} w^{[2]} &= w^{[2]} - \eta \cdot \frac{\partial E}{\partial w^{[2]}} = \begin{bmatrix} 1 & 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} -8.51848503 & -8.51848503 \end{bmatrix} \\ &= \begin{bmatrix} 1.8518485 & 1.8518485 \end{bmatrix} \end{aligned}$$

After updating weights, we'll update biases:

$$\frac{\partial E}{\partial b^{[1]}} = \delta^{[1]} \cdot \frac{\partial z^{[1]T}}{\partial b^{[1]}} = \left[ \sum_{i=1}^N \delta^{[1]} \right] \cdot 1 = \left[ \sum_{i=1}^N \delta^{[1]} \right] = \begin{bmatrix} -0.08518485 \\ -0.08518485 \end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[2]}} = \delta^{[2]} \cdot \frac{\partial z^{[2]T}}{\partial b^{[2]}} = \left[ \sum_{i=1}^N \delta^{[2]} \right] \cdot 1 = \left[ \sum_{i=1}^N \delta^{[2]} \right] = [-0.7012589]$$

$$b^{[1]} = b^{[1]} - \eta \cdot \frac{\partial E}{\partial b^{[1]}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} -0.08518485 \\ -0.08518485 \end{bmatrix} = \begin{bmatrix} 1.08518485 \\ 1.08518485 \end{bmatrix}$$

$$b^{[2]} = b^{[2]} - \eta \cdot \frac{\partial E}{\partial b^{[2]}} = [1] - 0.1 \cdot [-0.7012589] = [1.7012589]$$

## 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?

Calling the `print_regressor` method for each regressor shows us not only the MAE, but also the number of iterations required for each of the MLP regressors to converge. In this case, the number of iterations required for  $MLP_1$  ( $MLP$  with early stopping) to converge was 452, while  $MLP_2$  ( $MLP$  without early stopping) required only 77.

### 7. What can be motivating the unexpected differences on the number of iterations? Hypothesize 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.

With MLPRegressor's early stopping implementation, the training stops after a given number of epochs have passed without the validation score improving by at least a given tolerance (in order to avoid overfitting). The validation set, with MLPRegressor's



shuffle parameter set by default to True, may contain samples from both the training and test sets: with differing training sets (which forcefully seems like the case here), the training process may converge at a different amount of epochs; it's also likely that a training set with a lesser amount of samples may take more epochs to converge, as the algorithm will have to go through the entire dataset more times to reach the "same amount of samples seen" in order to better fit the data.

The **different number of iterations** could, then, be associated with the differing training sets used to train each regressor.

Regarding performance,  $MLP_1$  seems to perform better than  $MLP_2$ , with a lower (and thus better) MAE, as stated in question 4.'s answer. Although both regressors end up converging in training,  $MLP_1$  has the advantage of stopping right where the validation score starts to stagnate/decrease: this means that the regressor ends up not overfitting the data, as it would if it kept training for more epochs until reaching "regular" convergence, like  $MLP_2$  (which appears to be a bit more overfitted, in comparison to  $MLP_1$ ).

## Appendix

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from scipy.io.arff import loadarff
6 from sklearn.linear_model import Ridge
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import mean_absolute_error
9 from sklearn.neural_network import MLPRegressor
10 sns.set_style('darkgrid')
11
12 # Load data
13 data = loadarff('data/kin8nm.arff')
14 df = pd.DataFrame(data[0])
15
16 X = df.drop('y', axis=1).values
17 y = df['y'].values
18
19 X_train, X_test, y_train, y_test = train_test_split(
20     X, y,
21     test_size=0.3,
22     random_state=0
23 )
24
25 def predict(regressor):
26     regressor.fit(X_train, y_train)
27     return regressor.predict(X_test)
28
29 def plot_regressors_residues():
30     """Utilized for answering question 2."""
31     predictions = []
32     descriptions = []
33     for description, regressor in regressors.items():
34         predictions.append(predict(regressor))
35         descriptions.append(description)
36
37     fig, (ax1, ax2) = plt.subplots(2, figsize=(8, 12))
38
39     sns.histplot(
40         data=pd.DataFrame({
41             description: np.abs(y_test - prediction)
42             for description, prediction in zip(descriptions, predictions)
43         }),
44         bins=30,
45         ax=ax1,
46         multiple='dodge',
47     )
48     ax1.set_title('Residues histogram for each regressor')
49     ax1.set_xlabel('Residue')
50     ax1.set_ylabel('Count')
51     ax1.legend(descriptions)
52
53     sns.boxplot(
54         data=pd.DataFrame({
```

```

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()

```

