Aprendizagem 2024/25

Homework 3 – Group 98

(João Pinho - ist1102765, Tiago Sousa - ist1110691)

Part I: Pen and paper

1. Linear Regressor on transformed feature space using OLS

Applying the $\phi(y_1, y_2) = y_1 y_2$ transform to our 2D space, we get the following 3D transformed space:

D	y_1	y_2	$\phi(y_1, y_2)$	y_{num}
x_1	1	1	1	1.25
x_2	1	3	3	7.0
x_3	3	2	6	2.7
x_4	3	3	9	3.2
x_5	2	4	8	5.5

From this table we get the matrices we'll use to calculate the regressor's weights:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 6 \\ 1 & 9 \\ 1 & 8 \end{bmatrix} \qquad Z = \begin{bmatrix} 1.25 & 7.0 & 2.7 & 3.2 & 5.5 \end{bmatrix}$$

The Ordinary Least Squares estimator is a weight vector that minimizes the MSE(Mean square error) loss of a predictor. This means that it is the weight vector that minimizes

$$E(w) = \frac{1}{2} \sum_{i=1}^{n} (z_i - w^T * x_i)^2$$

In order to calculate that weight, the following formula is used:

$$W = (X^T * X)^{-1} * X^T * Z$$

Thus, our weight vector becomes:

$$W = \left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 6 & 9 & 8 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 6 \\ 1 & 9 \\ 1 & 8 \end{bmatrix} \right)^{-1} * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 6 & 9 & 8 \end{bmatrix} * \begin{bmatrix} 1.25 & 7.0 & 2.7 & 3.2 & 5.5 \end{bmatrix}$$

Using numpy's np.dot() to aid in calculation, we see that the matrix $X^T * X$ is not invertible. Therefore, we used the numpy function np.linalg.pinv(), which calculates the pseudo-inverse. Inserting the matrices into numpy and running the following code, we got the weight vector:

```
import numpy as np
X = np.array([[1,1],[1,3],[1,6],[1,9],[1,8]])
Z = np.array([1.25,7.0,2.7,3.2,5.5])
W = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(Z)
```

$$W = \begin{bmatrix} 3.31592 & 0.11371 \end{bmatrix}$$

2. Ridge regression

   For a Ridge regression, the error we need to minimize has a regularization term, and the formulas for E(w) and W become:

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(z_i - w^T * x_i)^2 + \frac{\lambda}{2} * \|w\|^2$$
$$W = (X^T * X + \lambda * I)^{-1} * X^T * Z$$

   where I represents the Identity matrix.

$$W = \left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 6 & 9 & 8 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 6 \\ 1 & 9 \\ 1 & 8 \end{bmatrix} + \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right)^{-1} * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 6 & 9 & 8 \end{bmatrix} * \begin{bmatrix} 1.25 & 7.0 & 2.7 & 3.2 & 5.5 \end{bmatrix}$$

   Similarly to the previous exercise, we used numpy to aid in calculation, and the code was:

```
import numpy as np
X = np.array([[1,1],[1,3],[1,6],[1,9],[1,8]])
Z = np.array([1.25,7.0,2.7,3.2,5.5])
I = np.eye(X.shape[1])
lambda = 1
W = np.linalg.inv(X.T.dot(X) + lambda * I).dot(X.T).dot(Z)
```

$$W = \begin{bmatrix} 1.81808 & 0.32375 \end{bmatrix}$$

Compared to the previous weights, these are significantly smaller, which is due to the regularization that penalizes bigger coefficients. Additionally, there could be some noise in the training set that gets corrected by the ridge regressor, whereas it gets learnt by the OLS. It seems that the OLS might have overfit the training set, as the bigger magnitude of the OLS can lead to more variance in its predictions and less generalization ability.
The regularization of the Ridge regressor creates a more robust and stable model, which should have a better performance on unseen data.

3. Comparing the models

Our models' predictions are given by:

$$\hat{Z} = X * W$$

Considering the new testing dataset:

D	y_1	y_2	$\phi(y_1, y_2)$	y_{num}
x_6	2	2	4	0.7
x_7	1	2	2	1.1
x_8	5	1	5	2.2

Applied to our dataset, we get:

$$\hat{Z}_{OLS-train} = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 6 \\ 1 & 9 \\ 1 & 8 \end{bmatrix} * \begin{bmatrix} 3.31592 \\ 0.11371 \end{bmatrix} = \begin{bmatrix} 3.42964 \\ 3.65707 \\ 3.99823 \\ 4.33938 \\ 4.22566 \end{bmatrix}$$

$$\hat{Z}_{Ridge-train} = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 6 \\ 1 & 9 \\ 1 & 8 \end{bmatrix} * \begin{bmatrix} 1.81808 \\ 0.32375 \end{bmatrix} = \begin{bmatrix} 2.14184 \\ 2.78936 \\ 3.76063 \\ 4.73191 \\ 4.40815 \end{bmatrix}$$

$$\hat{Z}_{OLS-test} = \begin{bmatrix} 1 & 4 \\ 1 & 2 \\ 1 & 5 \end{bmatrix} * \begin{bmatrix} 3.31592 \\ 0.11371 \end{bmatrix} = \begin{bmatrix} 3.77079 \\ 3.54336 \\ 3.88451 \end{bmatrix}$$

$$\hat{Z}_{Ridge-test} = \begin{bmatrix} 1 & 4 \\ 1 & 2 \\ 1 & 5 \end{bmatrix} * \begin{bmatrix} 1.81808 \\ 0.32375 \end{bmatrix} = \begin{bmatrix} 3.11312 \\ 2.46560 \\ 3.43687 \end{bmatrix}$$

To calculate the RMSE, we use the formula:

$$RMSE(\hat{Z}, Z) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Z}_i - Z_i)^2}$$

Our target variables are:

$$Z_{train} = \begin{bmatrix} 1.25 & 7.0 & 2.7 & 3.2 & 5.5 \end{bmatrix} \qquad Z_{test} = \begin{bmatrix} 0.7 & 1.1 & 2.2 \end{bmatrix}$$

So our RMSEs are as follows:

$$RMSE_{OLS-train} = \sqrt{\frac{4.75085 + 11.17511 + 1.68540 + 1.29818 + 1.62393}{5}} = \sqrt{4.10669} = 2.02649$$

$$RMSE_{Ridge-train} = \sqrt{\frac{0.79538 + 17.72947 + 1.12495 + 2.34676 + 1.19212}{5}} = \sqrt{4.63774} = 2.15354$$

$$RMSE_{OLS-test} = \sqrt{\frac{9.42979 + 5.97002 + 2.83758}{3}} = \sqrt{6.07913} = 2.46558$$

$$RMSE_{Ridge-test} = \sqrt{\frac{5.82315 + 1.86487 + 1.52987}{3}} = \sqrt{3.07263} = 1.75289$$

Analysing these results, we see that the training RMSE was similar for both models(2.02 and 2.15), with the OLS having slightly better results. Meanwhile, the testing was the other way round, as the Ridge regressor had significantly better RMSE than the OLS (1.75 compared to 2.46). This confirms our expectation that the ridge regressor had a better generalization ability, while the OLS was more prone to overfit and more sensitive to noise in the dataset.
Therefore, we can conclude that regularization helps increase the generalization ability of a regressor.

4. MLP with softmax in the output layer and cross-entropy loss to predict y_{class}
As $W^{[1]}$ and $W^{[2]}$ are 3×2 and 3×3 matrices, respectively, we can assume that layer 0 (input layer) has 2 nodes, layer 1 has 3 nodes and layer 2 (output layer) has 3 nodes.

Given that there is no activation in the hidden layer (layer 1), we'll define the activation functions for layer 1:

$$\phi^{[1]}(\mathbf{z}^{[1]}) = \mathbf{z}^{[1]}$$

The layer 2 is the last layer and hence the output layer, so the activation function is the softmax function:

$$\phi^{[2]}(z_i^{[2]}) = \text{softmax}\left(z_i^{[2]}\right) = \frac{e^{z_i^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}}$$

Let's start by predicting the output of observaiton $\mathbf{x}^{[0]} = \mathbf{x_1} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ using propagation.

For the hidden layer (1), we have:

$$x^{[1]} = \phi^{[1]}(\mathbf{z}^{[1]}) = \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x}^{[0]} + \mathbf{b}^{[1]} = \begin{pmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \\ 0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.3 \\ 0.3 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix}$$

For the output layer (2), let's first compute the net values of this layer:

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{x}^{[1]} + \mathbf{b}^{[2]} = \begin{pmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.7 \\ 1.3 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2.7 \\ 2.3 \\ 2 \end{pmatrix}$$

Now it's easier to compute the denominator of the softmax function:

$$\sum_{l=1}^{3} e^{z_l^{[2]}} = e^{z_1^{[2]}} + e^{z_2^{[2]}} + e^{z_3^{[2]}} = e^{2.7} + e^{2.3} + e^{2} \approx 32.243$$

Now let's compute the output (layer 2) signal:

- Prediction for class A:

$$x_1^{[2]} = \phi^{[2]}(z_1^{[2]}) = \frac{e^{z_1^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}} \approx \frac{e^{2.7}}{32.243} \approx 0.461$$

- Prediction for class B:

$$x_2^{[2]} = \phi^{[2]}(z_2^{[2]}) = \frac{e^{z_2^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}} \approx \frac{e^{2.3}}{32.243} \approx 0.309$$

- Prediction for class C:

$$x_3^{[2]} = \phi^{[2]}(z_3^{[2]}) = \frac{e^{z_3^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}} \approx \frac{e^2}{32.243} \approx 0.229$$

Hence, the output of observation $\mathbf{x_1}$ is $\mathbf{o} = \begin{pmatrix} 0.461 \\ 0.309 \\ 0.229 \end{pmatrix}$, while the training output of this observation is $\mathbf{t} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, because $y_{class}(\mathbf{x_1}) = B$.

We may now perform the stochastic gradient descent to the weights and the values.

We'll use the cross-entropy loss as an error function with only one ($n = 1$) observation (stochastic descent), hence:

$$E(\mathbf{W}) = -\sum_{i=1}^{n}\sum_{l=1}^{3} t_l^{(i)} log(o_l^{(i)}) = -\sum_{l=1}^{3} t_l log(o_l)$$

Let's start by obtaining the derivation formulas. First, let's find the partial derivatives of the output

$$\frac{\partial \mathbf{x}^{[p]}}{\partial \mathbf{W}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{W}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}\frac{\partial \mathbf{z}_j^{[p]}}{\partial \mathbf{W}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}\frac{\partial \mathbf{W}^{[p]}\mathbf{x}^{[p-1]} + \mathbf{b}^{[p]}}{\partial \mathbf{W}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}\mathbf{x}^{[p-1]T}$$

$$\frac{\partial \mathbf{x}^{[p]}}{\partial \mathbf{b}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{b}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}\frac{\partial \mathbf{z}_j^{[p]}}{\partial \mathbf{b}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}\frac{\partial \mathbf{W}^{[p]}\mathbf{x}^{[p-1]} + \mathbf{b}^{[p]}}{\partial \mathbf{b}^{[p]}} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}\mathbf{1} = \frac{\partial \phi^{[p]}(\mathbf{z}^{[p]})}{\partial \mathbf{z}^{[p]}}$$

We now have 2 cases. If we're considering the output layer (layer 2), the activation function is the softmax function, therefore:

$$\frac{\partial \phi^{[2]}(\mathbf{z}^{[2]})}{\partial \mathbf{z}^{[2]}} = \frac{\partial \frac{e^{\mathbf{z}^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}}}{\partial \mathbf{z}^{[2]}} = \frac{e^{\mathbf{z}^{[2]}}(\sum_{l=1}^{3} e^{z_l^{[2]}} - e^{\mathbf{z}^{[2]}})}{(\sum_{l=1}^{3} e^{z_l^{[2]}})^2} = \frac{e^{\mathbf{z}^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}}\left(\frac{\sum_{l=1}^{3} e^{z_l^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}} - \frac{e^{\mathbf{z}^{[2]}}}{\sum_{l=1}^{3} e^{z_l^{[2]}}}\right) =$$

$$= \phi^{[2]}(\mathbf{z}^{[2]})(\mathbf{I} - \phi^{[2]}(\mathbf{z}^{[2]})) = \mathbf{x}^{[2]}(\mathbf{I} - \mathbf{x}^{[2]}) = \mathbf{o}(\mathbf{t} - \mathbf{o})$$

Note that $\mathbf{I}$ is a vector with only zeroes except for one entry corresponding to the target class defined as 1, hence it corresponds to $\mathbf{t}$.

While in the hidden layer we have:

$$\frac{\partial \phi^{[1]}(\mathbf{z}^{[1]})}{\partial \mathbf{z}^{[1]}} = \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{z}^{[1]}} = \mathbf{1}$$

We may now finish the derivation of the update formulas:

$$\frac{\partial E}{\partial \mathbf{W}^{[2]}} = \frac{\partial E}{\partial \mathbf{x}^{[2]}}\frac{\partial \mathbf{x}^{[2]}}{\partial \mathbf{W}^{[2]}} = \frac{\partial \sum_{l=1}^{3} -t_l log(o_l)}{\partial \mathbf{o}}\frac{\partial \phi^{[2]}(\mathbf{z}^{[2]})}{\partial \mathbf{z}^{[2]}}\mathbf{x}^{[1]T} = \frac{\sum_{l=1}^{3} -t_l}{\mathbf{o}}\mathbf{o}(\mathbf{t}-\mathbf{o})\mathbf{x}^{[1]T} =$$
$$= (\mathbf{o}-\mathbf{t})\mathbf{x}^{[1]T}$$
$$\frac{\partial E}{\partial \mathbf{b}^{[2]}} = \frac{\partial E}{\partial \mathbf{x}^{[2]}}\frac{\partial \mathbf{x}^{[2]}}{\partial \mathbf{b}^{[2]}} = \frac{\partial \sum_{l=1}^{3} -t_l log(o_l)}{\partial \mathbf{o}}\frac{\partial \phi^{[2]}(\mathbf{z}^{[2]})}{\partial \mathbf{z}^{[2]}} = \frac{\sum_{l=1}^{3} -t_l}{\mathbf{o}}\mathbf{o}(\mathbf{t}-\mathbf{o}) = (\mathbf{o}-\mathbf{t})$$

(And in this case $\delta^{[2]} = (\mathbf{o}-\mathbf{t})$)

$$\frac{\partial E}{\partial \mathbf{W}^{[1]}} = \frac{\partial E}{\partial \mathbf{x}^{[1]}}\frac{\partial \mathbf{x}^{[1]}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \sum_{l=1}^{3} -t_l log(o_l)}{\partial \mathbf{x}^{[1]}}\frac{\partial \phi^{[1]}(\mathbf{z}^{[1]})}{\partial \mathbf{z}^{[1]}}\mathbf{x}^{[1]T} = \frac{\partial \sum_{l=1}^{3} -t_l log(o_l)}{\partial \mathbf{x}^{[1]}}\phi'^{[1]}(\mathbf{z}^{[1]})\mathbf{x}^{[1]T} =$$
$$= \ldots = \delta^{[1]}\mathbf{x}^{[1]T}$$
$$\frac{\partial E}{\partial \mathbf{b}^{[1]}} = \frac{\partial E}{\partial \mathbf{x}^{[1]}}\frac{\partial \mathbf{x}^{[1]}}{\partial \mathbf{b}^{[1]}} = \frac{\partial \sum_{l=1}^{3} -t_l log(o_l)}{\partial \mathbf{x}^{[1]}}\frac{\partial \phi^{[1]}(\mathbf{z}^{[1]})}{\partial \mathbf{z}^{[1]}} = \frac{\partial \sum_{l=1}^{3} -t_l log(o_l)}{\partial \mathbf{x}^{[1]}}\phi'^{[1]}(\mathbf{z}^{[1]}) =$$
$$= \ldots = \delta^{[1]}$$

Where

$$\delta^{[1]} = \mathbf{W}^{[2]^T}\delta^{[2]} \circ \phi'^{[1]}(z^{[1]})$$

We may now compute $\mathbf{W}^{[2]}$ after the update:

$$\mathbf{W}^{[2]}_{new} = \mathbf{W}^{[2]} - \eta\frac{\partial E}{\partial \mathbf{W}^{[2]}} = \mathbf{W}^{[2]} - \eta\delta^{[2]}\mathbf{x}^{[1]^T} = \mathbf{W}^{[2]} - \eta(\mathbf{o}-\mathbf{t})\mathbf{x}^{[1]^T} =$$

$$= \begin{pmatrix}1 & 2 & 2\\ 1 & 2 & 1\\ 1 & 1 & 1\end{pmatrix} - 0.1\left(\begin{pmatrix}0.461\\ 0.309\\ 0.229\end{pmatrix} - \begin{pmatrix}0\\ 1\\ 0\end{pmatrix}\right)\begin{pmatrix}0.3 & 0.3 & 0.4\end{pmatrix} = \begin{pmatrix}1 & 2 & 2\\ 1 & 2 & 1\\ 1 & 1 & 1\end{pmatrix} - \begin{pmatrix}0.0138 & 0.0138 & 0.0184\\ -0.0207 & -0.0207 & -0.0276\\ 0.0069 & 0.0069 & 0.0092\end{pmatrix} =$$

$$= \begin{pmatrix}0.986 & 1.986 & 1.982\\ 1.021 & 2.021 & 1.028\\ 0.993 & 0.993 & 0.991\end{pmatrix}$$

Let´s compute $b^{[2]}$ after the update:

$$\mathbf{b}^{[2]}_{new} = \mathbf{b}^{[2]} - \eta \frac{\partial E}{\partial \mathbf{b}^{[2]}} = \mathbf{b}^{[2]} - \eta \delta^{[2]} = \mathbf{b}^{[2]} - \eta(\mathbf{o} - \mathbf{t}) =$$

$$= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - 0.1 \left(\begin{pmatrix} 0.461 \\ 0.309 \\ 0.229 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0.0461 \\ -0.0691 \\ 0.0229 \end{pmatrix} = \begin{pmatrix} 0.954 \\ 1.069 \\ 0.977 \end{pmatrix}$$

Let's compute the $\delta^{[1]}$, that will make it easier to make the updates in the first layer:

$$\delta^{[1]} = \mathbf{W}^{[2]^T} \delta^{[2]} \circ \phi'^{[1]}(z^{[1]}) = \mathbf{W}^{[2]^T}(\mathbf{o} - \mathbf{t}) \circ \mathbf{1} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} \left(\begin{pmatrix} 0.461 \\ 0.309 \\ 0.229 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right) = \begin{pmatrix} 0 \\ -0.231 \\ 0.460 \end{pmatrix}$$

(Note that $\phi'^{[1]}$ is the identity function and thus its derivative is 1).

We may now compute $\mathbf{W}^{[1]}$ after the update:

$$\mathbf{W}^{[1]}_{new} = \mathbf{W}^{[1]} - \eta \frac{\partial E}{\partial \mathbf{W}^{[1]}} = \mathbf{W}^{[1]} - \eta \delta^{[1]} \mathbf{x}^{[0]^T} =$$

$$= \begin{pmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \\ 0.2 & 0.1 \end{pmatrix} - 0.1 \begin{pmatrix} 0 \\ -0.231 \\ 0.460 \end{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix} = \begin{pmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \\ 0.2 & 0.1 \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ -0.0231 & -0.0231 \\ 0.0460 & 0.0460 \end{pmatrix} =$$

$$= \begin{pmatrix} 0.1 & 0.1 \\ 0.1231 & 0.2231 \\ 0.154 & 0.054 \end{pmatrix}$$

And let's compute $\mathbf{b}^{[1]}$ after the update:

$$\mathbf{b}^{[1]}_{new} = \mathbf{b}^{[1]} - \eta \frac{\partial E}{\partial \mathbf{b}^{[1]}} = \mathbf{b}^{[1]} - \eta \delta^{[1]} =$$

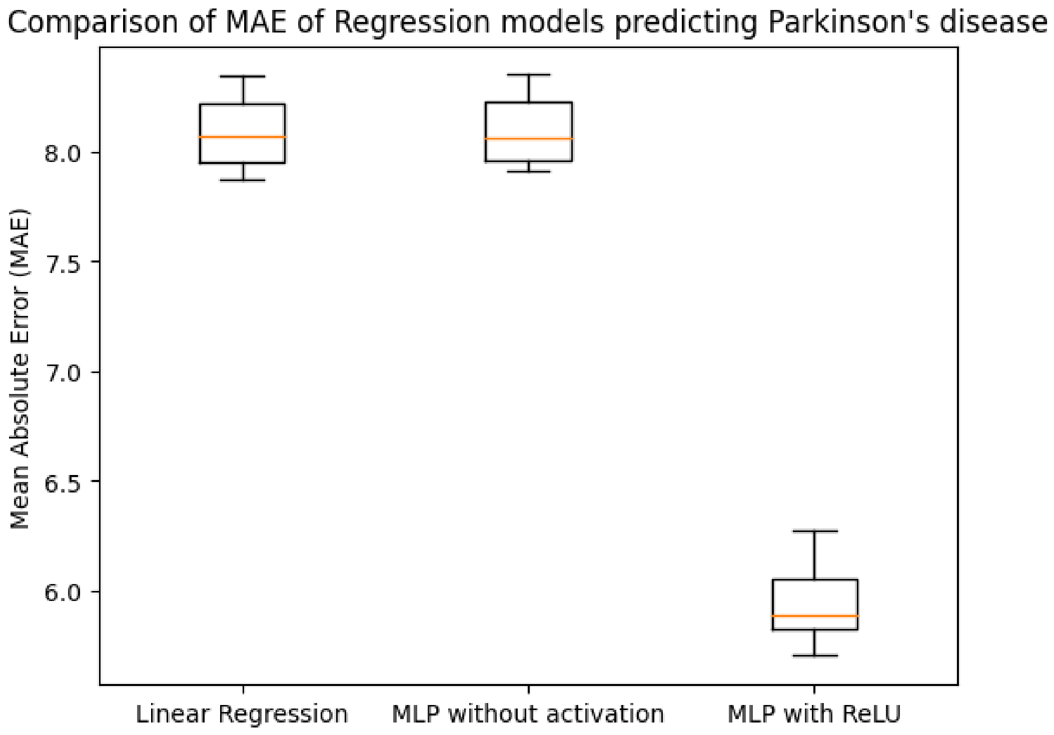
$$= \begin{pmatrix} 0.1 \\ 0 \\ 0.1 \end{pmatrix} - 0.1 \begin{pmatrix} 0 \\ -0.231 \\ 0.460 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0 \\ 0.1 \end{pmatrix} - \begin{pmatrix} 0 \\ -0.0231 \\ 0.0460 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.0231 \\ 0.0540 \end{pmatrix}$$

The final weights of the MLP after one stochastic gradient descent step will be:

$$\mathbf{W}^{[1]} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1231 & 0.2231 \\ 0.154 & 0.054 \end{bmatrix} \quad \mathbf{b}^{[1]} = \begin{bmatrix} 0.1 \\ 0.0231 \\ 0.0540 \end{bmatrix} \mathbf{W}^{[2]} = \begin{bmatrix} 0.986 & 1.986 & 1.982 \\ 1.021 & 2.021 & 1.028 \\ 0.993 & 0.993 & 0.991 \end{bmatrix} \quad \mathbf{b}^{[2]} = \begin{bmatrix} 0.954 \\ 1.069 \\ 0.977 \end{bmatrix}$$

Part 2: Programming

5. Comparison of MAE of a LinearRegression model, a MLP without activation and a MLP with ReLU activation



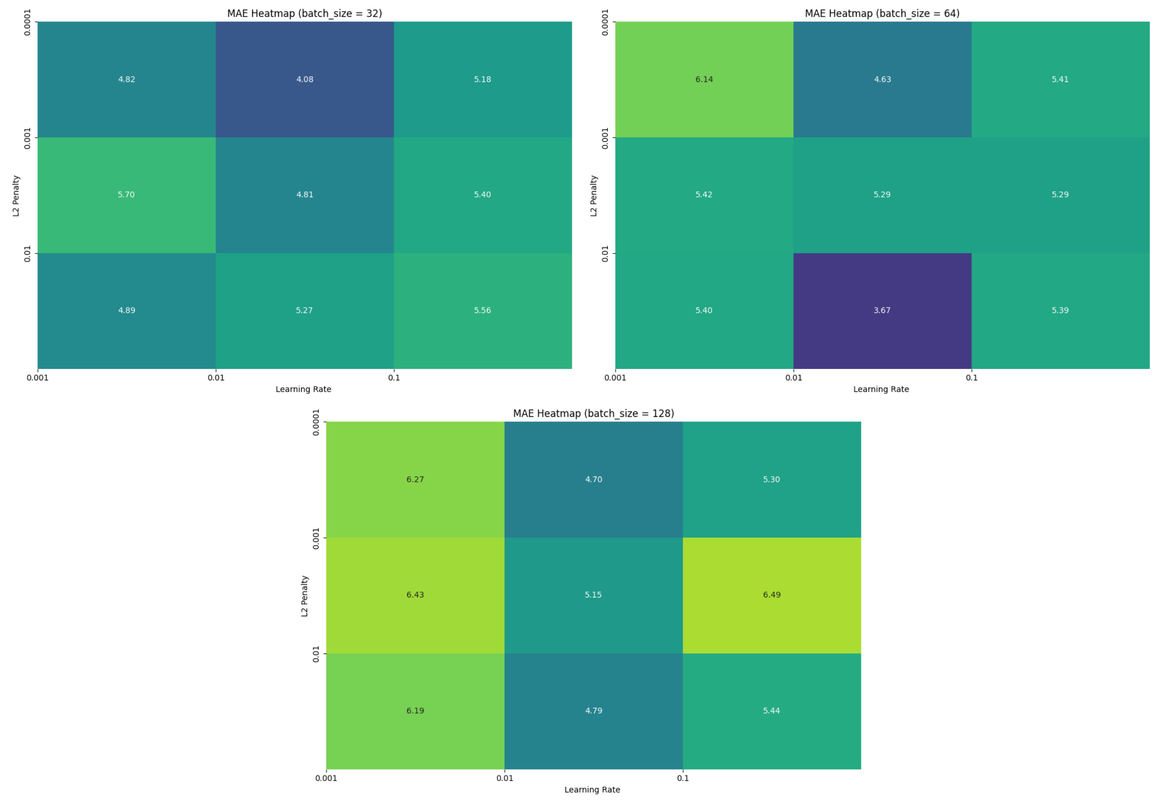
6. Discussing the importance of activation functions in MLPs

   The boxplot shows that the Linear Regression model and the MLP with no activation achieved very similar mean absolute errors of around 8, while the MLP with ReLU activation achieved a better MAE of around 6. The Linear Regression model and the MLP without activation achieved similar results probably because both are linear models: they only represent linear functions and are only able to devise linear patterns between the input and output.

   The MLP without activation is only able to represent the output as a linear function of the input because it only composes linear activation functions in each layer (in this case the identity function). The MLPs with nonlinear activation functions, such as ReLU, sigmoid or hyperbolic tangent, can represent nonlinear functions and thus devise nonlinear patterns in the data. The lower mean absolute error achieved by the MLP with ReLU in our dataset suggests that the dataset has nonlinear patterns.

7. Grid Search tuning an MLP

   In order to plot the MAE(mean absolute error) over all searched parameters, we plotted three heatmaps, one for each batch size, of the MAE score for that combination of parameters:



   These plots show us that the best combination of parameters was a learning rate of 0.01, L2 penalty of 0.01 and batch size of 64, which resulted in a MAE of 3.67.

   Some patterns are visually discernible, for example, the middle column has the best results in all 3 batch sizes, which means that the best learning rate is almost always 0.01. The learning rate also seems to be the most impactful parameter for the model's performance. The average MAE over each learning rate is: 4.71 for 0.01, 5.49 for 0.1 and 5.69 for 0.001.

   Also note that 4 out of the 5 worst results are in the third plot, when the batch size is 128, showing that a bigger batch size can lead to worse performances. It is also noteworthy that the batch size 32 was the most stable of all 3 batch sizes, as its worst was 5.7, compared to 6.14 and 6.49. This can indicate that a smaller batch size is less likely to lead to a bad result, even though it might not yield the best. The average MAE over each batch size is: 5.07 for 32, 5.18 for 64 and 5.64 for 128.

   Regarding the L2 penalty, both the top and bottom rows of each heatmap have mostly better MAEs than the middle one, suggesting that a good balance of regularization and learning rate is needed to obtain good results. The average MAE over each L2 penalty is: 5.17 for 0.0001, 5.17 for 0.01 and 5.55 for 0.001.