Group 37 – Enzo Nunes 106336, Tomás Barros 106588

Machine Learning – Homework3

Ex1 – Polynomial Regression

a)

Using Python and NumPy:

```python
import numpy as np

# Vector of observations
X = np.array([-1, 1, -1.2, 1.4, 1.9])

# Design matrix for a 3rd-degree polynomial
Phi = np.vstack([X**0, X**1, X**2, X**3]).T

print(Phi)
```

The design matrix is:

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1.2 & 1.44 & -1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.9 & 3.61 & 6.859 \end{bmatrix}$$

b)

Using Python and NumPy:

```python
# Target values array
t = np.array([-2, 3, -3, 0, -3])

# Compute the polynomial regression weights using the normal equation
# with the design matrix calculated previously and the target values
w = np.linalg.inv(Phi.T @ Phi) @ Phi.T @ t

print(w)
```

The weight vector is:

$$[\,3.02387284, 2.26101046, -2.63029446, -0.14838515]$$

c)

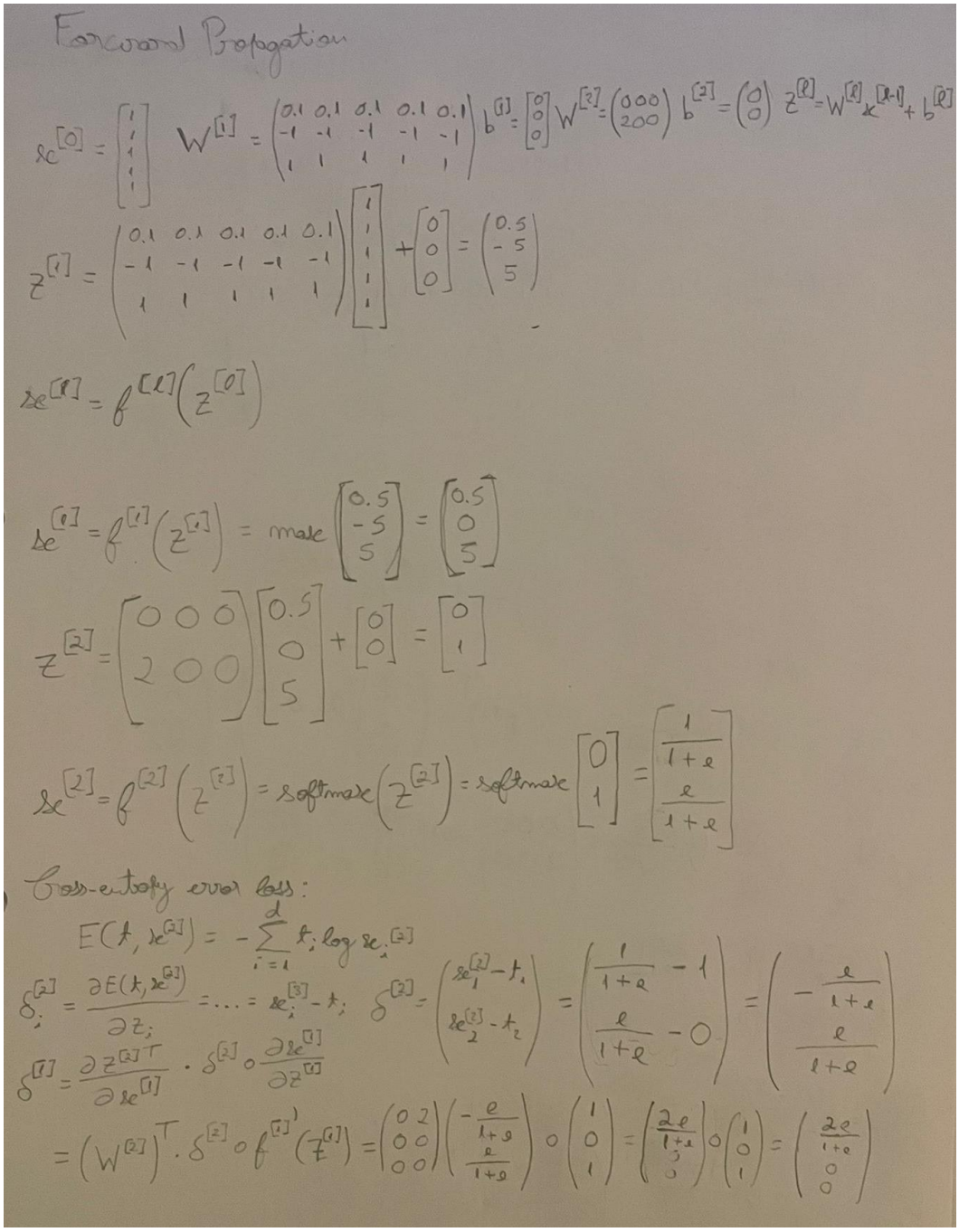
A closed-form solution exists for L2 regularization (Ridge regression) because the L2 penalty, which is the sum of the squared weights, results in a smooth and differentiable optimization problem. This allows the solution to be expressed using a modified normal equation:

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

Here, the regularization term ensures that the matrix is invertible. Since the problem remains linear in terms of the weights, it can be solved directly through matrix operations, providing a straightforward closed-form solution.

In contrast, L1 regularization (LASSO regression) penalizes the sum of the absolute values of the weights, leading to a non-differentiable objective function at zero. This non-linearity makes the optimization problem more complex, as the absolute value function does not allow for the same calculus-based solution approach used in Ridge regression. LASSO also tends to drive some weights to exactly zero, creating a sparse solution. Due to these properties, LASSO requires iterative methods like coordinate descent or gradient-based approaches rather than a closed-form solution.

Ex2 –Neural Network NN



Updates

$$\frac{\partial E}{\partial W^{[1]}} = \delta^{[1]} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}} = \delta^{[1]} \left(x^{[0]}\right)^T = \begin{pmatrix} \frac{2e}{1+e} \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2e}{1+e} & \frac{2e}{1+e} & \frac{2e}{1+e} & \frac{2e}{1+e} & \frac{2e}{1+e} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$W^{[1]}_{new} = W^{[1]}_{old} - \eta \frac{\partial E}{\partial W^{[1]}} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} - 0.1 \begin{pmatrix} \frac{2e}{1+e} & \frac{2e}{1+e} & \frac{2e}{1+e} & \frac{2e}{1+e} & \frac{2e}{1+e} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 0.1\left(1-\frac{2e}{1+e}\right) & 0.1\left(1-\frac{2e}{1+e}\right) & 0.1\left(1-\frac{2e}{1+e}\right) & 0.1\left(1-\frac{2e}{1+e}\right) & 0.1\left(1-\frac{2e}{1+e}\right) \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\frac{\partial E}{\partial b^{[1]}} = \delta^{[1]} \cdot \frac{\partial z^{[1]}}{\partial b^{[1]}} = \delta^{[1]}$$

$$b^{[1]}_{new} = b^{[1]}_{old} - \eta \frac{\partial z^{[1]}}{\partial b^{[1]}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - 0.1 \begin{pmatrix} \frac{2e}{1+e} \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -0.1\left(\frac{2e}{1+e}\right) \\ 0 \\ 0 \end{pmatrix}$$

$$\frac{\partial E}{\partial W^{[2]}} = \delta^{[2]} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}} = \delta^{[2]} \cdot \left(x^{[1]}\right)^T = \begin{pmatrix} -\frac{e}{1+e} \\ \frac{e}{1+e} \end{pmatrix} \begin{pmatrix} 0.5 & 0 & 5 \end{pmatrix} = \begin{pmatrix} -\frac{0.5e}{1+e} & 0 & -\frac{5e}{1+e} \\ \frac{0.5e}{1+e} & 0 & \frac{5e}{1+e} \end{pmatrix}$$

$$W^{[2]}_{new} = W^{[1]}_{old} - \eta \frac{\partial E}{\partial W^{[2]}} = \begin{pmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix} - 0.1 \begin{pmatrix} -\frac{0.5e}{1+e} & 0 & -\frac{5e}{1+e} \\ \frac{0.5e}{1+e} & 0 & \frac{5e}{1+e} \end{pmatrix} =$$

$$= \begin{pmatrix} \frac{0.05e}{1+e} & 0 & \frac{0.5e}{1+e} \\ 2-\frac{0.05e}{1+e} & 0 & \frac{0.5e}{1+e} \end{pmatrix}$$

$$\frac{\partial E}{\partial b^{[2]}} = \delta^{[2]} \cdot \frac{\partial z^{[2]}}{\partial b^{[2]}} = \delta^{[2]}$$

$$b^{[2]}_{new} = b^{[2]}_{old} - \eta \frac{\partial z^{[2]}}{\partial b^{[2]}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.1 \begin{pmatrix} -\frac{e}{1+e} \\ \frac{e}{1+e} \end{pmatrix} = \begin{pmatrix} \frac{0.1e}{1+e} \\ -\frac{0.1e}{1+e} \end{pmatrix}$$

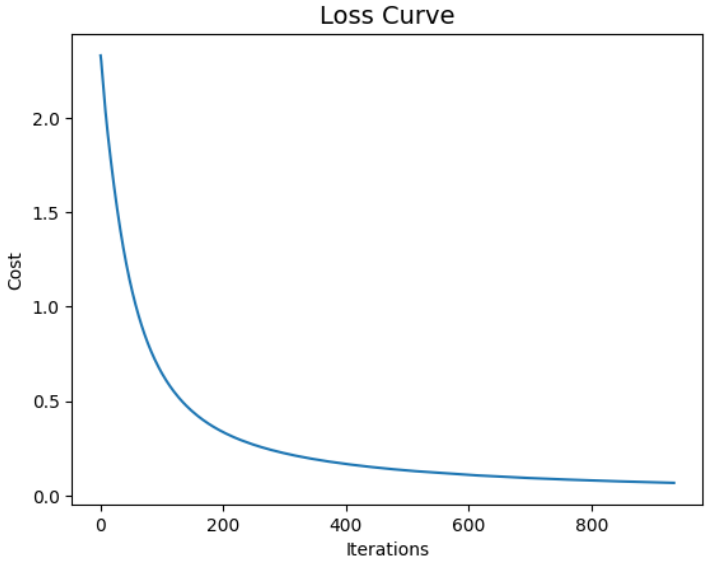
Ex3 – Software Experiments

- With the base code, Linear Regression yielded 96% accuracy, and the Multi-Layer Perceptron yielded 42.6% accuracy. It should be noted that the Neural Network reached the limit number of epochs before reaching convergence.
- However, by raising the number of neurons per layer, we can achieve way higher accuracies.

a)

After a few experiments, we concluded that we could increase the neural network's accuracy by increasing the number of neurons in each layer and decreasing the number of hidden layers. Using two layers of 50 neurons each, we achieved 96.55% accuracy. Using a single hidden layer of 110 neurons, we reached our best accuracy, of 97.33%.

Loss curve:



In conclusion, increasing the number of neurons in the hidden layers improved the neural network's accuracy, but adding more hidden layers decreased performance due to overfitting and optimization challenges.

b)

Using the identity function as the activation function, the accuracy decreased slightly, to 95.77%, which is worse than the Linear Regression's accuracy. This happens due to the identity function's linearity. We should not use this function as the activation function because it essentially turns the neural network into a linear model, regardless of how many hidden layers it has, limiting its ability to capture non-linear relationships in the data, which is essential for tasks like image classification.