Machine Learning Report

Homework III - Polynomial Regression and Neural Networks

ist1114964 - Axel Carapinha
ist1106565 - Martim Gordino

October 19, 2024

Contents

1 Polynomial Regression

1.1 Exercise a)

We have a non-linear transformation, so we apply it:

$$\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \\ 1 & x_5 & x_5^2 & x_5^3 \end{pmatrix}$$

Substituting the values of x:
- For $x_1 = -1$:

$$(1, -1, 1, -1)$$

- For $x_2 = 1$:

$$(1, 1, 1, 1)$$

- For $x_3 = -1.2$:

$$(1, -1.2, 1.44, -1.728)$$

- For $x_4 = 1.4$:

$$(1, 1.4, 1.96, 2.744)$$

- For $x_5 = 1.9$:

$$(1, 1.9, 3.61, 6.859)$$

So, the complete design matrix Φ is:

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

Noteworthy, the first column represents the bias.

1.2 Exercise b)

We calculate the pseudo-inverse $\Phi^\dagger$ using the formula:

$$\Phi^\dagger = \left(\Phi^T \Phi\right)^{-1} \Phi^T$$

Using the already calculated design matrix Φ:

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

$$\Phi^T\Phi = \begin{pmatrix} 5. & 2.1 & 9.01 & 7.875 \\ 2.1 & 9.01 & 7.875 & 20.9473 \\ 9.01 & 7.875 & 20.9473 & 27.65091 \\ 7.875 & 20.9473 & 27.65091 & 59.561401 \end{pmatrix}$$

$$\left(\Phi^T\Phi\right)^{-1} = \begin{pmatrix} 5.70227083 & -2.57484188 & -4.35147072 & 2.17175428 \\ -2.57484188 & 1.98153792 & 2.15163725 & -1.35533611 \\ -4.35147072 & 2.15163725 & 3.48654192 & -1.79997808 \\ 2.17175428 & -1.35533611 & -1.79997808 & 1.04193484 \end{pmatrix}$$

Given the target vector t:

$$t = \begin{pmatrix} -2 \\ 3 \\ -3 \\ 0 \\ -3 \end{pmatrix}$$

$$\Phi^\dagger = \left(\Phi^T\Phi\right)^{-1}\Phi^T$$

$$\Phi^\dagger = \begin{pmatrix} 1.75388771 & 0.94771252 & -1.22682816 & -0.47209666 & -0.00267541 \\ -1.04940644 & 0.20299718 & 0.48769105 & 0.69747794 & -0.33875972 \\ -1.21658797 & -0.51326963 & 1.19754707 & 0.55530376 & -0.02299322 \\ 0.68517748 & 0.05837494 & -0.59427422 & -0.39460409 & 0.24532589 \end{pmatrix}$$

Finally, the weights w are calculated as follows:

$$w = \Phi^\dagger \cdot t = \begin{pmatrix} 3.02387284 \\ 2.26101046 \\ -2.63029446 \\ -0.14838515 \end{pmatrix}$$

1.3 Exercise c)

This occurs because, unlike L2 regularization, L1 regularization (LASSO) leads to an optimization problem that is not differentiable at zero. As a result, a closed-form solution, which relies on the differentiability of the objective function, is not feasible.

Noteworthy, this is a problem because 0 is a possible value for the coefficient estimates, due to the absolute value penalty of L1 regularization (as opposed to the quadratic of L2) that encourages greater sparsity in these values.

2 Neural Network

We will need to calculate the derivates of the softmax function and derivative error, to use in the back propagation phase, and a later update phase.

In general, we have that

$$\text{softmax}\left(\begin{bmatrix} z_1 & z_2 & \cdots & z_d \end{bmatrix}\right)^\top = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix}^\top$$

$x_i = \frac{\exp(z)}{\sum_{k=1}^{d} \exp(z_k)}$ only on z_i

We can specify its derivative as follows: $\frac{\partial x_i}{\partial z_j} = \begin{cases} x_i(1 - x_i) & \text{if } i = j \\ -x_i x_j & \text{if } i \neq j \end{cases}$

Aanalysing the connection weights and biases, we can infer that there are 3 layers: an input layer (5 neurons), a hidden layer (3 neurons) and an output layer (the 2 final neurons).

Now, we can proceed with forward propagation:

$x^{[0]} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

$$z^{[1]} = W^{[1]} x^{[0]} + b^{[1]} = \begin{pmatrix} 0.5 \\ -5 \\ 5 \end{pmatrix}$$

Applying ReLU[1] activation function:

$$x^{[1]} = \text{ReLU}(z^{[1]}) = \begin{pmatrix} 0.5 \\ 0 \\ 5 \end{pmatrix}$$

$$z^{[2]} = W^{[2]} x^{[1]} + b^{[2]} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Applying softmax activation (as stated in the problem statement for the final layer):

$$x^{[2]} = \text{softmax}(z^{[2]}) = \begin{pmatrix} 0.2689 \\ 0.7311 \end{pmatrix}$$

Now, for the cross-entropy loss:

$$E(t, x) = -\sum_{i=1}^{d} t_i \log(x_i)$$

$$E(t, x^{[2]}) = -\left(1 \cdot \log(0.2689) + 0 \cdot \log(0.7311)\right) = 1.3133$$

[1]Rectified Linear Unit

Then, <u>backward propagation</u>.
At first, the deltas from the output and hidden layers (using the derivatives demonstrated in class):

$$\delta^{[2]} = x^{[2]} - t = \begin{pmatrix} -0.7311 \\ 0.7311 \end{pmatrix}$$

And after that calculating for the (single) hidden layer (using sign0, here represented as ReLU'):

$$\delta^{[1]} = W^{[2]^T} \delta^{[2]} \circ \text{ReLU}'(z^{[1]}) = \begin{pmatrix} -1.4622 \\ 0 \\ 0 \end{pmatrix}$$

For the last step, we <u>update the parameters</u>:

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

Starting with the first layer:

$$\frac{\partial E}{\partial W^{[2]}} = \delta^{[2]} \cdot \left(x^{[1]}\right)^T$$

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

$$\delta^{[2]} = \begin{pmatrix} -0.7311 \\ 0.7311 \end{pmatrix}, \quad x^{[1]} = \begin{pmatrix} 0.5 \\ 0 \\ 5 \end{pmatrix}$$

$$\frac{\partial E}{\partial W^{[2]}} = \begin{pmatrix} -0.7311 \\ 0.7311 \end{pmatrix} \cdot \begin{pmatrix} 0.5 & 0 & 5 \end{pmatrix} = \begin{pmatrix} -0.36555 & 0 & -3.6555 \\ 0.36555 & 0 & 3.6555 \end{pmatrix}$$

$$\frac{\partial E}{\partial b^{[2]}} = \begin{pmatrix} -0.7311 \\ 0.7311 \end{pmatrix}$$

$$W^{[2]} = W^{[2]} - \eta \frac{\partial E}{\partial W^{[2]}} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} -0.36555 & 0 & -3.6555 \\ 0.36555 & 0 & 3.6555 \end{pmatrix}$$

$$W^{[2]} = \begin{pmatrix} 0.03656 & 0 & 0.36555 \\ 1.96344 & 0 & -0.36555 \end{pmatrix}$$

$$b^{[2]} = b^{[2]} - \eta \frac{\partial E}{\partial b^{[2]}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} -0.7311 \\ 0.7311 \end{pmatrix}$$

$$b^{[2]} = \begin{pmatrix} 0.07311 \\ -0.07311 \end{pmatrix}$$

For the second one:

$$\frac{\partial E}{\partial W^{[1]}} = \delta^{[1]} \cdot \left(x^{[0]}\right)^T$$

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

$$\delta^{[1]} = \begin{pmatrix} -1.4622 \\ 0 \\ 0 \end{pmatrix}, \quad x^{[0]} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\frac{\partial E}{\partial W^{[1]}} = \begin{pmatrix} -1.4622 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -1.4622 & -1.4622 & -1.4622 & -1.4622 & -1.4622 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$W^{[1]} = W^{[1]} - \eta \frac{\partial E}{\partial W^{[1]}} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} -1.4622 & -1.4622 & -1.4622 & -1.4622 & -1.4622 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$W^{[1]} = \begin{pmatrix} -0.04621172 & -0.04621172 & -0.04621172 & -0.04621172 & -0.04621172 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$b^{[1]} = b^{[1]} - \eta \frac{\partial E}{\partial b^{[1]}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} -1.4622 \\ 0 \\ 0 \end{pmatrix}$$

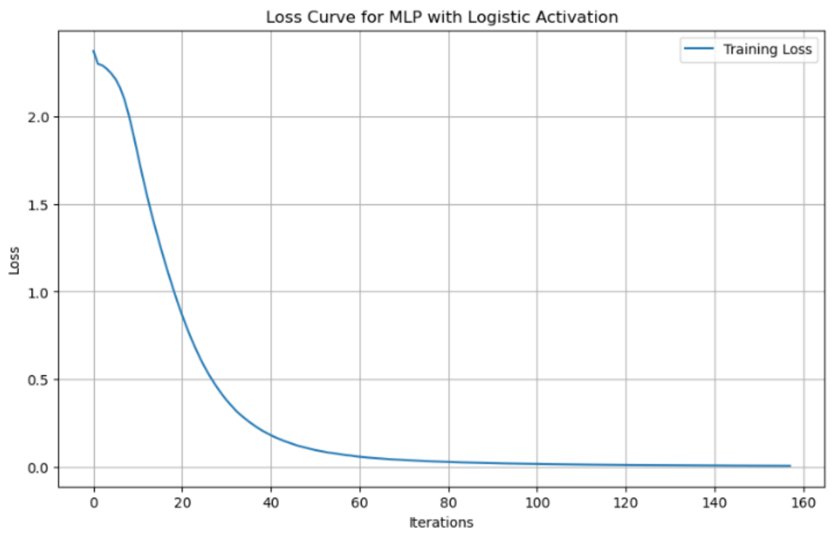
$$b^{[1]} = \begin{pmatrix} -0.14621172 \\ 0 \\ 0 \end{pmatrix}$$

3 Software Experiments

The used random state was 9.

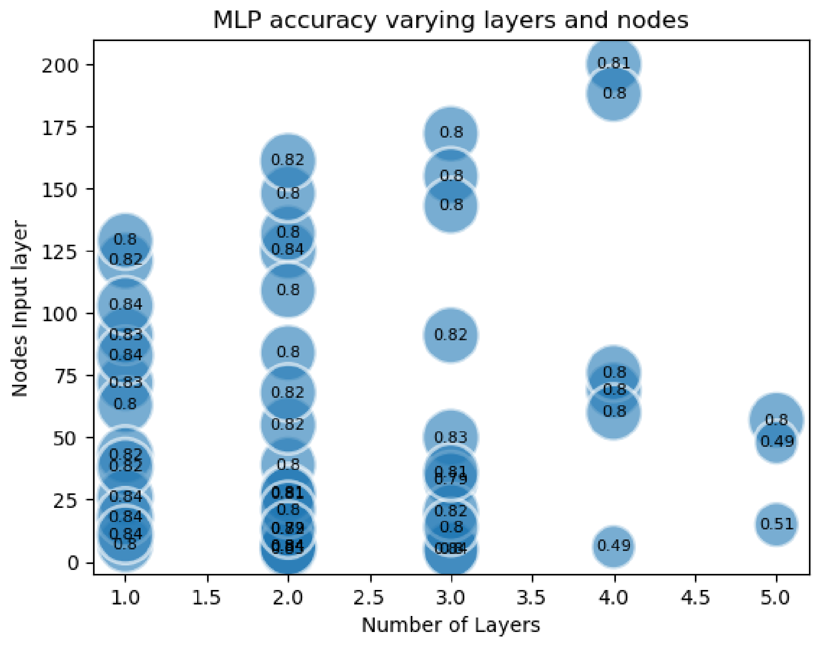
3.1 Exercise a)

The best values were achieved with 5 layers and 48 nodes per layer, with the following model's loss curve:



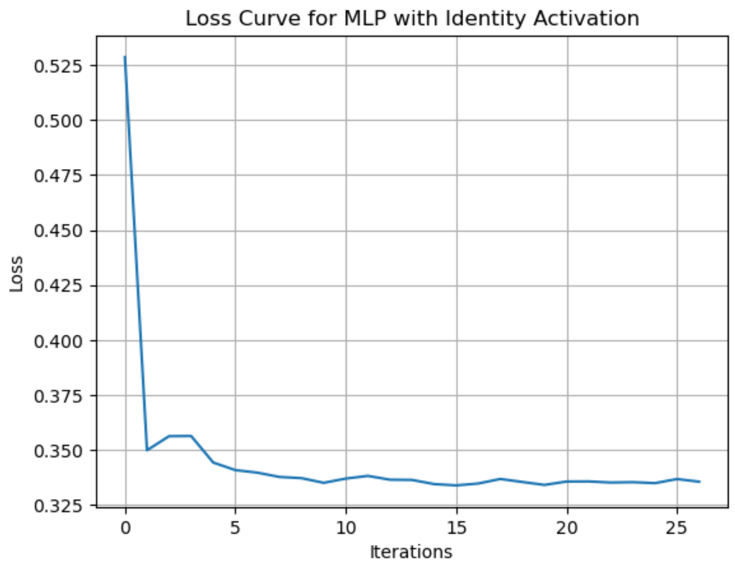
The model converges near 100 iterations, and the hamiltonian started to increase between 150 and 160 (overfiting), possibly explaining the stop of the model's training before the defined maximum of 2000 iterations (there can be more reasons abstracted by the used libraries).

Additionaly, here's a graphical representation of the model's accuracies, being generally higher with less layers and more nodes. However, sometimes more nodes appear to have lead to overfiting, when an intermediate value seems the best approach.



3.2 Exercise b)

The results were better at first (accurate faster), but remained in a certain interval (in a unstable manner), what can be explained by the linearity introduced by this function.



Using it created a linear relationship between data, something that is not the goal here, where the hidden layers commonly aim to generalize non-linear patterns.

Other functions, like tanh or sigmoid are better suited for this situation, not limiting the model's capability to capture the structure of the (small amount of) data, what can otherwise potentially lead to underfiting.