**I. Pen-and-paper**

1. Forward Propagation:

This part objective is to transform the input no output , for achieving this we apply recursively these functions

Applying forward propagation to the data given in the assignment, the following activation and net input vectors were obtained:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| (input) |  |  |  |  |  |  |
| 1 | 6 | 0.999988 | 3.76157 | 0.99892 | 0 | 0 |
| 1 | 1 | 0.761594 | 3.76157 | 0.99892 | 0 | 0 |
| 1 | 6 | 0.999988 |  | | | |
| 1 |  | | | | | |
| 1 |

Back Propagation:

After performing forward propagation, the stochastic gradient descent is applied in order get new weights and biases that better perform according to the loss function

The derivative needed to calculate SGD can be obtained using the chain rule:

Calculating the

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | |  |  | |  |
| 0 | 0 | 0.00216 | 0 | 0 | 1 | 0 | 0.999988 |
| 0 | 0 | 0 | 0.00216 | 0 | 0 | 1 | 0.761594 |
|  |  |  |  |  |  |  | 0.999988 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | |  |  | |  |
| -1 | 1 | 0 | -1 | 1 | 0 | 0.99892 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0.99892 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | |  |  | | |  |
| 1 | 1 |  | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0.419974 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |  | 0 | 0 | 0 | 1 | 1 |
|  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  |  |  | 1 |

Updates:

Now we have all the data needed to update the weights and biases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  | |  |  |
| -0.9989 | -0.9989 | 0.09989 | 0.09989 | -1 | 0.1 |
| 0.9989 | 0.9989 | -0.09989 | -0.09989 | 1 | -0.1 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | | | | |  |  |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | |  |  |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

1. This exercise is similar to the previous one, so the data calculated can be reused. The only results that were changed were:

Forward Propagation:

Back Propagation:

By applying these formulas we got:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| -2 | 0.25 | -0.25 | -0.5 |
| 0 | -0.25 | 0.25 | 0.5 |

Updates:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  | |  |  |
| -0.4995 | -0. 4995 | 0.04995 | 0.04995 | -0.5 | 0.05 |
| 0. 4995 | 0. 4995 | -0.04995 | -0.04995 | 0.5 | -0.05 |

**II. Programming and critical analysis**

1. The reasons for the observed differences are that with the presence of early stopping, an evaluation (with a validation split) is made each iteration, so the stopping criteria might involuntarily exclude crucial parts of the dataset. On the other hand, due to the stopping criteria, overfitting can be diminished, but in our specific case, it appears that the stopping criteria generates underfitting.  
     
   Chart, treemap chart

   Description automatically generated
2. Chart, box and whisker chart

   Description automatically generatedThe boxplots computed were the following:  
     
   The strategies we identified to minimize the observed error are:

* Increasing the value of regularization
* Lowering the learning rate will be more reliable, at the cost of execution time
* Changing the network complexity, namely its structure and parameters
* Regarding underfitting, increasing the capacity of the network, this problem can be easily addressed

**III. APPENDIX**

Code for question 2

data = arff.loadarff(“C:\\Users\\print\\Downloads\\kin8nm.arff”)

dataset = pandas.DataFrame(data[0])

inputs = dataset.drop(columns=[“y”]).values

outputs = dataset[“y”].values

predictions1 = np.ndarray(shape = (0,))

predictions2 = np.ndarray(shape = (0,))

actual = np.ndarray(shape = (0,))

k\_fold = Kfold(n\_splits=5, random\_state= 0, shuffle=True)

classifier1 = MLPRegressor(alpha = 20, activation = “relu”, hidden\_layer\_sizes=(3, 2), early\_stopping = False,  random\_state= 13, max\_iter = 1500)

classifier2 = MLPRegressor(alpha = 1e-5, activation = “relu”, hidden\_layer\_sizes=(3, 2), early\_stopping = False,  random\_state= 13, max\_iter = 1500)

for train, test in k\_fold.split(dataset):

    classifier1.fit(inputs[train], outputs[train])

    predicted1 = classifier1.predict(inputs[test])

    classifier2.fit(inputs[train], outputs[train])

    predicted2 = classifier2.predict(inputs[test])

    predictions1 = np.concatenate((predictions1,predicted1), axis= 0)

    predictions2 = np.concatenate((predictions2,predicted2), axis= 0)

    actual = np.concatenate((actual,outputs[test]), axis= 0)

residuals1 = actual – predictions1

residuals2 = actual – predictions2

residuals = {“With Regularization”: residuals1, “Without Regularization”: residuals2}

fig, ax = plt.subplots()

ax.boxplot(residuals.values())

ax.set\_xticklabels(residuals.keys())

plt.show()

Code for question 3

data = arff.loadarff("C:\\Users\\print\\Downloads\\kin8nm.arff")

dataset = pandas.DataFrame(data[0])

inputs = dataset.drop(columns=["y"]).values

outputs = dataset["y"].values

predictions1 = np.ndarray(shape = (0,))

predictions2 = np.ndarray(shape = (0,))

actual = np.ndarray(shape = (0,))

k\_fold = KFold(n\_splits=5, random\_state= 0, shuffle=True)

classifier1 = MLPRegressor(alpha = 20, activation = "relu", hidden\_layer\_sizes=(3, 2), early\_stopping = False,  random\_state= 13, max\_iter = 1500)

classifier2 = MLPRegressor(alpha = 1e-5, activation = "relu", hidden\_layer\_sizes=(3, 2), early\_stopping = False,  random\_state= 13, max\_iter = 1500)

for train, test in k\_fold.split(dataset):

    classifier1.fit(inputs[train], outputs[train])

    predicted1 = classifier1.predict(inputs[test])

    classifier2.fit(inputs[train], outputs[train])

    predicted2 = classifier2.predict(inputs[test])

    predictions1 = np.concatenate((predictions1,predicted1), axis= 0)

    predictions2 = np.concatenate((predictions2,predicted2), axis= 0)

    actual = np.concatenate((actual,outputs[test]), axis= 0)

residuals1 = actual - predictions1

residuals2 = actual - predictions2

residuals = {"With Regularization": residuals1, "Without Regularization": residuals2}

fig, ax = plt.subplots()

ax.boxplot(residuals.values())

ax.set\_xticklabels(residuals.keys())

plt.show()

**END**