**I. Pen-and-paper**

u[i]=ui; ∂= d; 𝛿=&; 𝜂=n

1. Forward Propagation:

Backward Propagation:

* Auxiliary Calculations:

* Updates:

**b)**

Taking advantage of some of the conclusions taken in 1.a):

Forward Propagation:

(Since only the output unit has a different activation function, only x\_3 is affected)

Backward Propagation:

* Auxiliary Calculations:
* Updates:

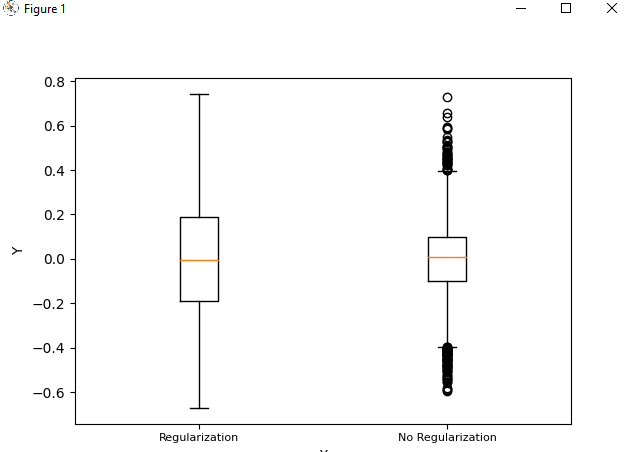
**II. Programming and critical analysis**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pred | | |
| True |  | N(benign) | P(malign) |
| N (benign) | 389 | 55 |
| P (malign) | 5 | 234 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pred | | |
| True |  | N(benign) | P(malign) |
| N (benign) | 1559 | 217 |
| P (malign) | 23 | 933 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pred | | |
| True |  | N(benign) | P(malign) |
| N (benign) | 1678 | 98 |
| P (malign) | 65 | 891 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pred | | |
| True |  | N(benign) | P(malign) |
| N (benign) | 419 | 25 |
| P (malign) | 18 | 221 |

1. Early Stopping (Test) No Early Stopping (Test)  
   Accuracy=623/683=91.22% Accuracy = 640/683=93.70%  
     
   Early Stopping (Train) No Early Stopping (Train)  
   Accuracy=2492/2732=91.22% Accuracy = 2569/2732=94.03%  
     
     
     
     
     
     
     
     
     
   Early stopping is generally used to prevent overfitting, as the model does not attempt to completely fit the training data and thus is more generalized:  
   - The training accuracy is higher than the testing accuracy with no early stopping because, by following through with every update, the model overfits itself to the training observations and considers either noise or variables unrelated to the output (there is no difference in training and testing accuracy for the opposite reason, with early-stopping).  
   - However, there is also higher accuracy (both training and testing) with no early stopping, because since there are smaller batches of data due to 5-fold cross-validation, it is useful to go through all the epochs to truly learn the model.
2. alpha=10 vs alpha=0  
   Strategies:

* Increase learning rate (usually lowering it would lessen noise contribution to predicted output, but since it is already relatively low at 𝜂=0.001, increasing it to about 𝜂=0.01 actually helps lower observed error)
* Clustering data into a smaller number of variables (lowers VC which makes accuracy higher)
* Change activation function/loss function combination (a better combination might lower noise contribution to predicted output)
* Increase layer size (by at least 1, but not many more than that) and number of layers (by at least 1, but not many more than that)

**III. APPENDIX**

**2)**

import numpy as np; from scipy.io import arff; import pandas as pd;

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import StratifiedKFold;

from sklearn.metrics import confusion\_matrix

tableVar = pd.DataFrame(arff.loadarff("breast.w.arff")[0]).dropna()

tableVar["Class"] = tableVar["Class"].apply(lambda y : y.decode("utf-8"))

x=[]

for i in range(len(tableVar)):x += [list(tableVar.iloc[i][0:9])]

x= np.array(x);y = list(tableVar["Class"]);y= np.array(y)

for el in range(len(y)):

    if y[el] == "benign":y[el] = 0

    else:y[el] = 1

d\_TN\_TP\_FN\_FP\_true\_test = {"TP": 0, "TN": 0, "FP": 0, "FN": 0}; d\_TN\_TP\_FN\_FP\_true\_train = {"TP": 0, "TN": 0, "FP": 0, "FN": 0}

d\_TN\_TP\_FN\_FP\_false\_test = {"TP": 0, "TN": 0, "FP": 0, "FN": 0}; d\_TN\_TP\_FN\_FP\_false\_train = {"TP": 0, "TN": 0, "FP": 0, "FN": 0}

def check\_stats\_test(\_y\_test, \_y\_pred\_test, dic\_in):

    cnf\_matrix = confusion\_matrix(\_y\_test, \_y\_pred\_test)

    \_FP = cnf\_matrix[0][1]; \_FN = cnf\_matrix[1][0]; \_TP = cnf\_matrix[1][1];

    \_TN = cnf\_matrix[0][0]; \_FP = \_FP.astype(float); \_FN = \_FN.astype(float);

    \_TP = \_TP.astype(float); \_TN = \_TN.astype(float);

    dic\_in["FP"] += \_FP; dic\_in["TP"] += \_TP

    dic\_in["FN"] += \_FN; dic\_in["TN"] += \_TN

    return dic\_in

skf = StratifiedKFold(n\_splits=5,random\_state=0, shuffle=True)

for train\_index, test\_index in skf.split(x, y):

    X\_train, X\_test = x[train\_index], x[test\_index]; y\_train, y\_test = y[train\_index], y[test\_index]

    clf\_early\_stop\_true = MLPClassifier(random\_state=0,early\_stopping=True, hidden\_layer\_sizes=[3,2]).fit(X\_train, y\_train)

    y\_pred\_test\_true = clf\_early\_stop\_true.predict(X\_test)

    y\_pred\_train\_true = clf\_early\_stop\_true.predict(X\_train)

    early\_stop\_true\_y\_test = check\_stats\_test(y\_test, y\_pred\_test\_true, d\_TN\_TP\_FN\_FP\_true\_test)

    early\_stop\_true\_y\_train = check\_stats\_test(y\_train, y\_pred\_train\_true, d\_TN\_TP\_FN\_FP\_true\_train)

    #======================================= False ===============================

    clf\_early\_stop\_false = MLPClassifier(hidden\_layer\_sizes=[3,2],random\_state=0,early\_stopping=False).fit(X\_train, y\_train)

    y\_pred\_test\_early\_False = clf\_early\_stop\_false.predict(X\_test)

    y\_pred\_train\_early\_False = clf\_early\_stop\_false.predict(X\_train)

    early\_stop\_false\_y\_test = check\_stats\_test(y\_test, y\_pred\_test\_early\_False, d\_TN\_TP\_FN\_FP\_false\_test)

    early\_stop\_false\_y\_train = check\_stats\_test(y\_train, y\_pred\_train\_early\_False, d\_TN\_TP\_FN\_FP\_false\_train)

def dict\_to\_confusion\_matrix(dict\_conf\_mat):

    M = np.matrix([[dict\_conf\_mat["TN"], dict\_conf\_mat["FP"]],

                   [dict\_conf\_mat["FN"], dict\_conf\_mat["TP"]]])

    return M

print("Confusion Matrix Early Stop True Test\n", dict\_to\_confusion\_matrix(d\_TN\_TP\_FN\_FP\_true\_test))

print("Confusion Matrix Early Stop True Train\n", dict\_to\_confusion\_matrix(d\_TN\_TP\_FN\_FP\_true\_train))

print("Confusion Matrix Early Stop False Test\n", dict\_to\_confusion\_matrix(d\_TN\_TP\_FN\_FP\_false\_test))

print("Confusion Matrix Early Stop False Train\n", dict\_to\_confusion\_matrix(d\_TN\_TP\_FN\_FP\_false\_train))

**3)**

import numpy as np; from scipy.io import arff; import pandas as pd

from sklearn.model\_selection import KFold

from sklearn.neural\_network import MLPRegressor

import matplotlib.pyplot as plt

tableVar = pd.DataFrame(arff.loadarff("kin8nm.arff")[0]).dropna()

x=[]

for i in range(len(tableVar)):x += [list(tableVar.iloc[i][0:8])]

x= np.array(x); y = list(tableVar["y"]); y= np.array(y)

y\_test\_all = []; y\_test\_pred\_all\_regularization = []

y\_test\_all\_noRegular = []; y\_test\_pred\_all\_noRegular = []

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

for train\_index, test\_index in skf.split(x, y):

    X\_train, X\_test = x[train\_index], x[test\_index]; y\_train, y\_test = y[train\_index], y[test\_index]

    y\_test\_all += list(y\_test)

    reg\_all = MLPRegressor(alpha= 10, random\_state=0, hidden\_layer\_sizes=[3,2]).fit(X\_train, y\_train)

    y\_test\_pred\_all\_regularization += list(reg\_all.predict(X\_test))

    Noreg\_all = MLPRegressor(alpha= 0, random\_state=0, hidden\_layer\_sizes=[3,2]).fit(X\_train, y\_train)

    y\_test\_pred\_all\_noRegular += list(Noreg\_all.predict(X\_test))

def compute\_residues(list\_y\_test, list\_pred\_test):

    list\_residues = []

    sum\_modules = 0

    for i in range(len(list\_y\_test)):

        list\_residues += [list\_y\_test[i] - list\_pred\_test[i]]

        sum\_modules += np.abs(list\_y\_test[i] - list\_pred\_test[i])

    return list\_residues, sum\_modules

data1, sum1 = compute\_residues(y\_test\_all, y\_test\_pred\_all\_regularization)

data2, sum2 = compute\_residues(y\_test\_all, y\_test\_pred\_all\_noRegular)

print("sum1= ", sum1)

print("sum2= ", sum2)

data = [data1, data2]; random\_dists = ["Regularization", "No Regularization"]

fig, ax = plt.subplots()

ax.boxplot(data); ax.set\_xlabel('X'); ax.set\_ylabel('Y')

ax.set\_xticklabels(np.repeat(random\_dists,1),rotation=0, fontsize=8)

plt.show()

**END**