ASS3_FFANN

April 30, 2022

Machine Learning FFANN

ASSIGNMENT 3

21L-7289 - Saad Ather 21L-7285 -Zeshan Asif

---

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

0.0.1 Iris Data set

[117]:
```
data=pd.read_csv("Iris.csv",header=0)
data.head()
```

[117]:
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

[118]:
```
## Data pre-processing
df2 = data.loc[:
 ↪,['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Species']]
feat = df2.iloc[:,:-1].values
label = df2.iloc[:,-1:].values

#Encoding Categorical Variable Species
from sklearn.preprocessing import LabelEncoder
LE1 = LabelEncoder()
label = np.array(LE1.fit_transform(label[:,-1]))
Label = pd.get_dummies(label).values

#spliting data into test and train set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(feat,Label,test_size=30)
```

```python
#Normalising the inputs
sc = StandardScaler()
sc.fit(x_train)
X_train=sc.transform(x_train)
X_test = sc.transform(x_test)
```

[119]:
```python
error = []
class FeedForwardANN:

    ## Class inatialization
    def __init__(self,inputlayer,hiddenlayer,outputlayer):
        np.random.seed(1)
        self.numb_of_Iperceptron = inputlayer
        self.numb_of_Hperceptron = hiddenlayer[0]
        self.numb_of_Operceptron = outputlayer
        self.numb_of_Hlayer = hiddenlayer[1]
        self.W1 = np.random.randn(inputlayer,16)
        self.W2 = np.random.randn(16,outputlayer)
        self.B1 = np.zeros((1,16))
        self.B2 = np.zeros((1,outputlayer))

    ## Activation function
    def sigmoid(self, X):
        return 1.0/(1.0 + np.exp(-X))

    ## Forward Propogate
    def forward_propagate(self,X):
        y_hat =[]
        self.Z1 = np.matmul(X,self.W1) + self.B1
        self.A1 = self.sigmoid(self.Z1)
        self.Z2 = np.matmul(self.A1, self.W2) + self.B2
        self.A2 = self.sigmoid(self.Z2)
        y_hat.append(self.A2)
        return y_hat

    ## Trainig based on Forward Propogate
    def train(self, inputs, targets, epochs, learning_rate):
        for i in range(epochs):
            sum_errors = 0
            # iterate through all the training data
            for j, input in enumerate(inputs):
                target = targets[j]
                # activate the network!
                output = self.forward_propagate(input)
                error.append(np.subtract(target, output))
                # tracking of the MSE
                sum_errors += self._mse(target, output)

        print('\n\nMean Square Error is : {} \n'.format(sum_errors))


    ## Error/Cost function
    def _mse(self, target, output):
        return np.average((target - output) ** 2)
```

[120]:
```
#pred = []
FFANN = FeedForwardANN(len(df2.columns)-1,[16,1],3)#len(np.unique(label)))
FFANN.train(X_train, Label, 50, 0.2)
pred = FFANN.forward_propagate(X_test)
print('Predicted outcome on test data in %age form of each perceptron in Output␣
 ↪Layer \n\n {} '.format(pred[0]*100))
```

```


Mean Square Error is : 43.97812338696865

Predicted outcome on test data in %age form of each perceptron in Output Layer

 [[82.43405402 79.0069015  47.6712108 ]
 [90.84463936 24.09507642 86.86804162]
 [80.7701501  79.90624925 33.48863806]
 [86.54854466 52.40941483 79.2939006 ]
 [89.04978484 34.63721903 72.60231012]
 [82.31934998 78.40497227 43.14459615]
 [91.96949461 12.6703995  82.13653671]
 [81.32458627 79.43654269 19.58721038]
 [90.64278301 26.2781325  78.84189661]
 [90.1774125  28.64363144 47.20237966]
 [87.24112599 42.47747139 42.94904319]
 [91.99514431 21.54697297 76.78830138]
 [91.23141379 35.73969289 66.06160982]
 [91.88669292 15.21630386 80.62157659]
 [90.63232872 27.89300453 66.62184069]
 [81.24296121 81.99138674 23.56212051]
 [86.39186329 52.63214112 71.74950383]
 [87.03934017 53.80770925 73.95255016]
 [88.67685    39.78793121 48.68343987]
 [83.43016852 86.05887372 14.96282616]
 [90.03294496 32.29083684 76.13721232]
 [90.77434371 29.54370946 73.21364701]
 [87.88342921 38.98961053 65.63873371]
 [81.18242538 83.03511251 27.81309786]
 [88.03858082 42.56290913 45.76294709]
 [86.04449273 43.7854114  63.52663081]
 [89.02486811 31.32948557 63.53607898]
 [81.45809676 72.25528769 44.89417661]
 [89.73408917 39.02170068 34.00232506]
 [80.73238782 75.83999468 21.24719612]]
```

[121]:
```
predict =[]
for i in range(len(pred[0])):
    max = np.max(pred[0][i])
    if (max == pred[0][i][0]):
        predict.append('Iris-virginicia')
    if (max == pred[0][i][1]):
        predict.append('Iris-Setosa')
    if (max == pred[0][i][2]):
        predict.append('Iris-versicolor')
```

[122]:
```
actual = []
for i in range(len(y_test)):
    maxact = np.max(y_test[i])
    if (maxact == y_test[i][0]):
        actual.append('Iris-Setosa')
    if (maxact == y_test[i][1]):
        actual.append('Iris-versicolor')
    if (maxact == y_test[i][2]):
        actual.append('Iris-virginica')
```

[123]:
```
predicted_Label = pd.DataFrame(np.array(predict),columns=['Predicted Label'])
actual_Label = pd.DataFrame(np.array(actual),columns=['Actual Label'])
feat_col = pd.
 ↪DataFrame(data=x_test,columns=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm

Final = pd.concat([feat_col,actual_Label,predicted_Label],axis='columns')
```

[124]:
```
Final
```

[124]:
```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm     Actual Label  \
0            5.8           4.0            1.2           0.2      Iris-Setosa
1            7.7           2.6            6.9           2.3   Iris-virginica
2            5.4           3.7            1.5           0.2      Iris-Setosa
3            7.0           3.2            4.7           1.4  Iris-versicolor
4            6.3           3.3            4.7           1.6  Iris-versicolor
5            5.7           3.8            1.7           0.3      Iris-Setosa
6            6.3           3.3            6.0           2.5   Iris-virginica
7            4.8           3.4            1.6           0.2      Iris-Setosa
8            6.5           3.2            5.1           2.0   Iris-virginica
9            5.4           3.0            4.5           1.5  Iris-versicolor
```

```
10             5.5           2.6            4.4           1.2  Iris-versicolor
11             6.4           2.8            5.6           2.2   Iris-virginica
12             6.3           2.5            5.0           1.9   Iris-virginica
13             6.3           3.4            5.6           2.4   Iris-virginica
14             6.0           3.0            4.8           1.8   Iris-virginica
15             5.1           3.5            1.4           0.2      Iris-Setosa
16             6.6           2.9            4.6           1.3  Iris-versicolor
17             6.7           3.1            4.4           1.4  Iris-versicolor
18             5.6           3.0            4.1           1.3  Iris-versicolor
19             4.9           3.0            1.4           0.2      Iris-Setosa
20             6.7           2.5            5.8           1.8   Iris-virginica
21             6.4           2.7            5.3           1.9   Iris-virginica
22             6.1           3.0            4.6           1.4  Iris-versicolor
23             5.4           3.4            1.7           0.2      Iris-Setosa
24             5.6           2.7            4.2           1.3  Iris-versicolor
25             6.1           2.8            4.7           1.2  Iris-versicolor
26             6.0           2.7            5.1           1.6  Iris-versicolor
27             5.5           4.2            1.4           0.2      Iris-Setosa
28             5.2           2.7            3.9           1.4  Iris-versicolor
29             4.8           3.4            1.9           0.2      Iris-Setosa

    Predicted Label
0   Iris-virginicia
1   Iris-virginicia
2   Iris-virginicia
3   Iris-virginicia
4   Iris-virginicia
5   Iris-virginicia
6   Iris-virginicia
7   Iris-virginicia
8   Iris-virginicia
9   Iris-virginicia
10  Iris-virginicia
11  Iris-virginicia
12  Iris-virginicia
13  Iris-virginicia
14  Iris-virginicia
15      Iris-Setosa
16  Iris-virginicia
17  Iris-virginicia
18  Iris-virginicia
19      Iris-Setosa
20  Iris-virginicia
21  Iris-virginicia
22  Iris-virginicia
23      Iris-Setosa
24  Iris-virginicia
```

```
25   Iris-virginicia
26   Iris-virginicia
27   Iris-virginicia
28   Iris-virginicia
29   Iris-virginicia
```

0.1 Conclusion

Our model evaluated the results successufully but with the RMS error of 43.3, model precdicted the results on class label Iris virginica and Iris setosa with some level of accuracy but failed to accurately predict the results on class label Iris Versicolor. If back propagation is applied it is then possible to incresse the accuracy of our model to further extend which will be left for now for future aspect.

0.2 MODEL EVALUATION ON MNIST DATASET

[88]:
```
mnist = pd.read_csv('mnist.csv')
```

[89]:
```
mnist.head()
```

[89]:
```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  …  28x19  28x20  \
0      5    0    0    0    0    0    0    0    0    0  …      0      0
1      0    0    0    0    0    0    0    0    0    0  …      0      0
2      4    0    0    0    0    0    0    0    0    0  …      0      0
3      1    0    0    0    0    0    0    0    0    0  …      0      0
4      9    0    0    0    0    0    0    0    0    0  …      0      0

   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0

[5 rows x 785 columns]
```

[106]:
```
## Data pre-processing
feat = mnist.iloc[:,1:].values
label = mnist.iloc[:,0:1].values
```

[107]:
```
#Encoding Categorical Variable Species
from sklearn.preprocessing import LabelEncoder
LE1 = LabelEncoder()
label = np.array(LE1.fit_transform(label[:,-1]))
Label = pd.get_dummies(label).values

#spliting data into test and train set
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(feat,Label,test_size=30)

#Normalising the inputs
#sc = StandardScaler()
#sc.fit(x_train)
#X_train=sc.transform(x_train)
#X_test = sc.transform(x_test)
```

[ ]:

```
```

[103]:

```
#pred = []
FFANN = FeedForwardANN(len(mnist.columns)-1,[16,1],len(np.unique(label)))
FFANN.train(x_train, Label, 50, 0.2)
pred = FFANN.forward_propagate(x_test)
#print('Predicted outcome on test data, of each perceptron in Output Layer \n\n␣
↪{} '.format(pred))
```

```
C:\Users\RAPTOR\anaconda3\envs\tf\lib\site-packages\ipykernel_launcher.py:18:
RuntimeWarning: overflow encountered in exp



Mean Square Error is : 24516.35904751916
```

0.3 Conclusion

Our model didnt not gave result near to any accuaray and completly failed the predicted outcomes with the mean square error of very large value, when the number of perceptrons in hidden layers are 16. Maybe it will give better results on increasing the hidden layers and number of perceptrons in each layer.

[116]:

```
print('Predicted outcome in %age form of each perceptron in Output Layer is␣
↪\n\n {} '.format(pred[0]*100))
```

```
Predicted outcome in %age form of each perceptron in Output Layer is

 [[39.1929935  28.15324436 11.05363905 60.27991878 56.60665906 93.56906745
  30.79561788 82.27871608 27.53939616 93.98709199]
 [51.26401712 93.91617505 38.49164682 48.82665193 11.21821458 86.19038437
   6.30134539 80.63615676 72.37169717 94.06650867]
 [72.38433555 37.06765673  5.19553948 85.39310136 37.39292154 85.98110432
   1.38111653 89.94477695 90.53299027 97.82285245]
 [51.30805512 63.53109595 73.28146696 31.29022292 16.74377582 56.60819676
   4.90439629 50.53226618 58.15753391 98.43375677]
 [ 1.81206847  8.31778491  3.82242993 94.69180056 23.79897988 89.64367045
  28.20647904 94.4491751  52.01465684 68.87759191]
```

```
[77.14875714 54.24713921 52.55451788 75.2983812  25.70472487 69.14115532
  1.66320829 89.33201765 80.74519436 92.55644826]
[26.25257031 86.0565812   9.95687469 49.03066105  7.18411148 82.43337858
 43.24675298 83.22356507 65.7318538  76.16240447]
[73.68271247 63.68881606  5.83713217 68.8004372   4.02395521 65.02248679
 29.20033018 89.90904106 84.51627141 88.47638018]
[22.56921985 56.95176146 31.67778842 93.55156039 26.0825942  91.48712555
 19.51128201 92.40767496 54.27076774 87.32512996]
[35.31671964 90.1172737   4.41735927 96.12876474 22.16865207 90.77377024
  4.35185171 85.46759244 75.31462718 80.7854871 ]
[32.47073766 62.18733691 32.66112998 95.27658342 18.96364699 61.2126909
  9.85575228 73.13922349 71.11368576 85.80701644]
[80.81390761 85.42102535  7.61973945 56.79699774 16.30393872 88.49925178
  2.05872763 92.9633514  92.23458822 85.79078268]
[25.1265175  12.75734949 27.7189359  64.69705524  9.76714067 55.18133536
 47.68871535 91.4238726  57.68468532 79.69654578]
[20.49754825 32.786445   21.67832642 94.2175809  23.26864073 92.2870435
  3.86537125 93.90440759 74.82238574 95.97811611]
[97.81297796 54.45135129 36.65956688 67.60136195 34.71616061 52.55943076
  0.63823285 84.40153058 86.22549294 98.84272367]
[62.1384595  22.87459241 14.59937661 90.33658386 17.76684315 94.81433531
  4.05579679 91.03246066 75.36507806 99.68678252]
[93.5908089  91.17639808  3.66811401 58.10909509 46.31733888 59.17965888
  1.16499362 73.33296772 84.3939924  73.65107374]
[95.15426184 34.21041398 51.3153333  52.31230414 14.33583044 88.59207806
  5.89224239 92.6187343  73.79112386 99.69499924]
[88.46576504 63.49207357  3.11846836 51.49051861 16.51290536 92.50704712
  5.66185199 92.59474324 80.28168942 98.01647891]
[97.15587624 69.02684055 65.76576187 47.34575127 33.13745323 67.4101888
  4.15022554 86.25490031 70.92131919 97.32930742]
[33.00477009 17.18442262 16.61932376 79.04339506 11.97085496 37.07204442
 45.25173322 87.93585266 58.04207639 72.67721585]
[11.98600704 23.83785932  5.15703326 98.76023028 23.9263225  62.78129
  3.85230902 79.79541511 70.35497781 81.87171246]
[59.82006534 28.35765528 34.05961744 85.42945785 14.19180957 61.90561249
 17.28429943 91.73622762 58.75387015 90.14021929]
[46.10672652 90.3922781   5.82593065 96.35825356 13.15321704 69.74430449
  4.03785999 67.76562501 83.22005698 84.45599266]
[74.49015949 62.91762233  2.13390728 92.46001142 12.26759033 96.39075132
  1.60084249 88.68469535 86.03949765 99.65160005]
[81.5504808  38.87526028 25.02168029 87.61398102 27.07268468 54.5473447
  0.25027427 87.83506202 91.49884711 96.68236063]
[70.97631454 66.70761004 62.53254335 35.88543613 19.2098196  91.47682643
 53.69217717 96.8496495  60.56170732 79.26594102]
[87.94577663 91.63965335 11.35308896  9.53464529 38.16041989 96.34584585
  8.26888456 77.10521096 55.45279959 97.83510213]
[84.46676939 11.48550842 20.51190536 73.43421207 23.1891393  80.00890339
  0.79461921 90.93498868 78.25852597 99.71712094]
```

```
[96.70923028 76.29732082 66.29116683 49.53322662 14.34332326 59.15706213
 10.94335337 75.4489867  70.09442883 98.20732679]]
```

```
[ ]:
```