In [1]:
```
import arff
import random
import csv
import pandas as pd
import scipy as sc
import numpy as np
import copy
import numpy as np
import math
import sklearn
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:
```
from mlp import *
```

In [3]:
```
linsep_raw = arff.load(open('linsep2nonorigin.arff'))
linsep_raw
```

Out[3]:
```
{'description': '',
 'relation': 'linSep2nonorigin',
 'attributes': [('a1', 'REAL'), ('a2', 'REAL'), ('class', ['0', '1'])],
 'data': [[-0.4, 0.3, '1'],
  [-0.3, 0.8, '1'],
  [-0.2, 0.3, '1'],
  [-0.1, 0.9, '1'],
  [-0.1, 0.1, '0'],
  [0.0, -0.2, '0'],
  [0.1, 0.2, '0'],
  [0.2, -0.2, '0']]}
```

In [4]:
```
linsep_values = []
linsep_labels = []
for entry in linsep_raw['data']:
    linsep_values.append(entry[0:-1])
    linsep_labels.append((entry[-1]))

print(linsep_labels)
linsep_values
```

```
['1', '1', '1', '1', '0', '0', '0', '0']
```

Out[4]:
```
[[-0.4, 0.3],
 [-0.3, 0.8],
 [-0.2, 0.3],
 [-0.1, 0.9],
 [-0.1, 0.1],
 [0.0, -0.2],
 [0.1, 0.2],
 [0.2, -0.2]]
```

In [5]:
```
linsep_MLP = MLP(hidden_nodes=7, no_improvement_break=50, max_iterations=500)
linsep_MLP.fit(linsep_values, linsep_labels)
```

In [6]:
```
linsep_MLP.rmse[-1]
```

Out[6]:
```
0.3251689042154294
```

In [7]:
```
linsep_MLP.training_rmse[-1]
```

Out[7]:
```
0.17929620072301924
```

In [8]:
```
linsep_MLP.best_rmse
```

Out[8]:
```
0.32421871148249015
```

It looks like the training RMSE is about half of the rmse on the test set

In [9]:
```
output_classes = linsep_MLP.predict(linsep_values)
```

In [10]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [11]:
```
print(classification_report(linsep_labels, output_classes))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         4

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8
```

In [12]:
```
print(confusion_matrix(linsep_labels, output_classes))
```

```
[[4 0]
 [0 4]]
```

In [13]:
```
linsep_MLP.best_weights
```

Out[13]:
```
[array([[ 0.11148551,  2.13100393,  1.5143493 ,  1.90857385,  0.67508502,
         -1.4871642 , -0.8652406 ],
        [ 1.82942441, -1.76130441, -1.23038736, -1.30935416,  0.69211759,
          2.74994393,  2.3877805 ],
        [ 0.09855578,  0.72663742,  0.49069241,  0.5793337 ,  0.57347255,
         -0.84282389, -0.57307099]]),
 array([[-0.96458417,  1.06325498],
        [ 1.97896554, -2.07503459],
        [ 1.33870275, -1.54011628],
        [ 1.75495498, -1.49122819],
        [ 0.10190068,  0.11472172],
        [-2.35977586,  2.43908526],
        [-1.52428792,  2.17427577],
        [ 0.34928714, -0.93793717]])]
```

Well, I guess when you only have 4 in each of 2 classes, it is pretty easy to get them correct

In [14]:
```
# now I am going to do this again with the specifications given in the homework
# except I didn't update the learning rate. I need to do that.
# I also didn't make it possible to use all zeros for starting weights

linsep_MLP = MLP(hidden_nodes=4, no_improvement_break=100, max_iterations=10, step_size
linsep_MLP.fit(linsep_values, linsep_labels)  #or just values instead of scaled_values
```

In [15]:
```
linsep_MLP.weight_matrices
```

Out[15]:
```
[array([[ 9.25998130e-05,  9.25998130e-05,  9.25998130e-05,
          9.25998130e-05],
        [-5.66370570e-04, -5.66370570e-04, -5.66370570e-04,
         -5.66370570e-04],
        [-2.03710193e-03, -2.03710193e-03, -2.03710193e-03,
         -2.03710193e-03]]),
```

```
array([[-0.00202777,  0.00202777],
       [-0.00202777,  0.00202777],
       [-0.00202777,  0.00202777],
       [-0.00202777,  0.00202777],
       [-0.0041204 ,  0.0041204 ]])]
```

In [16]:
```
# this is what was recorded in the homeworks
[1.050641719962177451e-02, 1.050641719962177451e-02, 1.050641719962177451e-02, 1.050641
[2.148777913098560283e-02, -1.050641719962178491e-02, -1.050641719962178491e-02, -1.050
[-1.050641719962178491e-02, -2.148777913098558895e-02, -1.814943182760951840e-04, 1.574


[-7.882184463621048215e-03, -1.814943182760951840e-04]
[1.574684975438346004e-03, -7.882184463621048215e-03]
[-1.814943182760951840e-04, 1.574684975438346004e-03]
[-7.882184463621048215e-03, -1.814943182760951840e-04]
[1.574684975438346004e-03, -7.882184463621048215e-03]
```

Out[16]:
```
[0.001574684975438346, -0.007882184463621048]
```

Mine always splits the input so that it can use part of it for training and the other part for test, but my results don't look anything like the what the homework has

In [17]:
```
linsep_MLP.best_weights
```

Out[17]:
```
[array([[ 9.25998130e-05,  9.25998130e-05,  9.25998130e-05,
          9.25998130e-05],
        [-5.66370570e-04, -5.66370570e-04, -5.66370570e-04,
         -5.66370570e-04],
        [-2.03710193e-03, -2.03710193e-03, -2.03710193e-03,
         -2.03710193e-03]]),
 array([[-0.00202777,  0.00202777],
        [-0.00202777,  0.00202777],
        [-0.00202777,  0.00202777],
        [-0.00202777,  0.00202777],
        [-0.0041204 ,  0.0041204 ]])]
```

In [18]:
```
linsep_MLP.best_epoch
```

Out[18]:
```
10
```

In [19]:
```
linsep_MLP.rmse
```

Out[19]:
```
[0.5000092474847885,
 0.5000082721464579,
 0.5000074458146871,
 0.5000067433474011,
 0.5000061441546593,
 0.5000056313332366,
 0.5000051909708021,
 0.5000048115858652,
 0.5000044836764735,
 0.5000041993560841]
```

Well, not starting with all zeros appears to have changed this a whole ton. Oh well.

In [20]:
```
# Now I want to try out the sklearn and see what I get
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.neural_network import MLPClassifier
import numpy as np
import matplotlib.pyplot as plt
```

In [25]:
```
clf = MLPClassifier(shuffle=False, hidden_layer_sizes=(4), activation='logistic', max_i
```

```
/home/dsant/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_per
ceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1) reached
and the optimization hasn't converged yet.
  warnings.warn(
```

In [26]:
```
clf.predict(linsep_values)
```

Out[26]:
```
array(['0', '0', '0', '0', '0', '0', '0', '0'], dtype='<U1')
```

In [27]:
```
clf.best_loss_
```

Out[27]:
```
0.7050818323670163
```

In [28]:
```
clf.coefs_
```

Out[28]:
```
[array([[ 0.63565085,  0.53380833, -0.57033358,  0.35316601],
        [-0.61215549,  0.25486784,  0.55987672, -0.37600693]]),
 array([[-0.21660287],
        [-0.12942012],
        [ 0.28889482],
        [-0.23126928]])]
```

It looks like this didn't start with initial weights of zero. Nor sure how to make it do that. Oh well.

In [ ]:
```

```

In [61]:
```
banknote_raw = arff.load(open('data_banknote_authentication.arff'))
#banknote_raw
```

In [60]:
```
banknote_values = []
banknote_labels = []
for entry in banknote_raw['data']:
    banknote_values.append(entry[0:-1])
    banknote_labels.append((entry[-1]))

#print(banknote_labels)
#banknote_values
```

In [31]:
```
banknote_MLP = MLP(hidden_nodes=4, no_improvement_break=50, max_iterations=10, step_siz
banknote_MLP.fit(banknote_values, banknote_labels)  #or just values instead of scaled_v
```

In [32]:
```
banknote_MLP.best_weights
```

Out[32]:
```
[array([[ 1.72646327,  1.25672837,  2.04746753,  1.9100961 ],
        [ 1.12489484,  0.83004856,  1.38612976,  1.07766463],
        [ 1.17194602,  0.79095454,  1.83396084,  1.12893832],
        [ 0.23330998,  0.25591873, -0.10774083,  0.19011937],
        [-1.93360367, -1.03700479, -2.17480297, -1.57100632]]),
 array([[ 2.35785354, -1.88345969],
        [ 1.24299337, -1.04846519],
        [ 2.28540574, -2.78842378],
        [ 1.92960427, -2.11313382],
        [-4.13140164,  4.24973826]])]
```

I don't have any notes on what these weights are supposed to be, but I would be willing to bet mine are way off because I started with random weights instead of zeros.

I am curious as to how well this one did.

In [33]:
```
banknote_MLP.weight_matrices
```

Out[33]:
```
[array([[ 1.72646327,  1.25672837,  2.04746753,  1.9100961 ],
        [ 1.12489484,  0.83004856,  1.38612976,  1.07766463],
        [ 1.17194602,  0.79095454,  1.83396084,  1.12893832],
        [ 0.23330998,  0.25591873, -0.10774083,  0.19011937],
        [-1.93360367, -1.03700479, -2.17480297, -1.57100632]]),
 array([[ 2.35785354, -1.88345969],
        [ 1.24299337, -1.04846519],
        [ 2.28540574, -2.78842378],
        [ 1.92960427, -2.11313382],
        [-4.13140164,  4.24973826]])]
```

In [34]:
```
banknote_MLP.best_epoch
```

Out[34]:
```
10
```

In [35]:
```
banknote_outputs = banknote_MLP.predict(banknote_values)
```

In [36]:
```
print(classification_report(banknote_labels, banknote_outputs))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       762
           1       0.98      1.00      0.99       610

    accuracy                           0.99      1372
   macro avg       0.99      0.99      0.99      1372
weighted avg       0.99      0.99      0.99      1372
```

In [37]:
```
print(confusion_matrix(banknote_labels, banknote_outputs))
```

```
[[752  10]
 [  3 607]]
```

Well, the accuracy is certainly something I can live with. I am surprised it was so good after only 10 epochs.

In [38]:
```
banknote_MLP.rmse
```

Out[38]:
```
[0.2015709651590803,
 0.1487187706655097,
 0.12812643120483908,
 0.1181637506482045,
 0.11405614033061764,
 0.11265910408811221,
 0.11243695962545622,
 0.11209806351498572,
 0.11086833946901156,
 0.10792504366226126]
```

In [39]:
```
banknote_MLP.training_rmse
```

Out[39]:
```
[0.2002780340525674,
 0.1364135692111928,
 0.11201568009142907,
 0.10072341124852455,
 0.09543612027109746,
 0.0928873083802931,
```

```
 0.09166827647492065,
 0.09076187798654597,
 0.08943985007168123,
 0.08717173350403289]
```

In [59]:
```
#banknote_MLP.confidence_scores(banknote_values)
```

In [58]:
```
#banknote_MLP.output_z_values
```

In [42]:
```
for i, thing in enumerate(banknote_MLP.output_z_values):
    if max(thing) < 0.7 or min(thing) > 0.3:
        print(i)
        print(thing)
```

```
911
[0.4862065884679286, 0.5709976802928265]
925
[0.5115268408943882, 0.5372699812592264]
979
[0.3879438977448633, 0.6567297401854217]
1040
[0.37373977379386925, 0.6675184319197549]
1094
[0.4915269362985097, 0.5674091794454563]
1101
[0.6863859093547825, 0.35329374907131644]
1345
[0.6229707863213976, 0.42652391709135573]
```

This can tell me which ones I have very little confidence in their scores

Now I want to see how well the scikitlearn one does. I suspect the accuracy should be similar (possibly better)

In [43]:
```
clf = MLPClassifier(shuffle=False, hidden_layer_sizes=(4), activation='logistic', max_i
```

```
/home/dsant/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_per
ceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reache
d and the optimization hasn't converged yet.
  warnings.warn(
```

In [44]:
```
banknote_outputs = clf.predict(banknote_values)
print(classification_report(banknote_labels, banknote_outputs))
print()
print(confusion_matrix(banknote_labels, banknote_outputs))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       762
           1       0.98      0.99      0.99       610

    accuracy                           0.99      1372
   macro avg       0.99      0.99      0.99      1372
weighted avg       0.99      0.99      0.99      1372


[[752  10]
 [  7 603]]
```

It is slightly worse, actually. I wonder if I made it have a slower learning rate somehow.

Now I want to look at how these perform on the iris dataset.

In [45]:
```
iris_raw = arff.load(open('iris.arff'))
#iris_raw
```

In [57]:
```
values = []
labels = []
for entry in iris_raw['data']:
    values.append(entry[0:-1])
    labels.append((entry[-1]))

#print(labels)
#values
```

In [47]:
```
irisMLP = MLP(hidden_nodes=15, no_improvement_break=50, max_iterations=1500, porportion
irisMLP.fit(values, labels)
```

In [48]:
```
print(irisMLP.best_epoch)
print(irisMLP.best_rmse)
```

```
234
0.14519523443795226
```

In [49]:
```
irisMLP.training_rmse[234]
```

Out[49]:
```
0.10361827673328458
```

In [50]:
```
irisClf = MLPClassifier(shuffle=False, hidden_layer_sizes=(15), activation='logistic',
```

In [51]:
```
irisClf.best_loss_ # well, that appears tons better than mine
```

Out[51]:
```
0.04695283763545288
```

In [52]:
```
irisMLP.best_weights
```

Out[52]:
```
[array([[ 0.08461092,  0.6029573 ,  0.18617144, -0.94202542,  0.55899757,
          0.78528109, -3.4057539 ,  0.17853381,  0.35099107,  0.89162626,
          1.05252572,  0.7717855 , -3.70957732,  0.40892237,  0.39288989],
        [ 0.81964395,  0.30641933,  0.99668975, -1.52618591,  0.57302524,
          0.69273804, -3.9985131 ,  0.77344162,  0.42065574,  0.65181752,
          0.15344702,  0.03565185, -3.1156933 ,  0.50520184,  0.78645097],
        [ 0.81365611,  0.47584711,  0.82323195,  3.19581097,  0.91427732,
          0.95511316,  6.05720997,  0.93695447,  0.5212013 ,  0.2029875 ,
          0.24733885,  0.8554517 ,  5.93962894,  0.48857071,  0.44129393],
        [ 0.45982762,  0.87524267,  0.85492237,  1.84332699,  0.37208984,
          0.9750989 ,  4.12346505,  0.13197064,  0.0400179 ,  0.41206517,
          0.95140962,  0.08580854,  4.04676201,  0.05338121,  0.3010534 ],
        [ 0.52864568,  0.84685627,  0.33252961, -0.22372295,  0.40027809,
          0.13902317, -3.54995485,  0.85494306,  0.61193944,  0.47726747,
          0.05577181,  0.85914191, -3.54019461,  0.78885171,  0.60568935]]),
 array([[ 0.38051326,  0.14552746, -0.58648086],
        [ 0.22418111, -0.3707056 , -0.0793547 ],
        [ 0.56496386,  0.25807171, -0.92326771],
        [-6.60803308,  6.86995235,  2.66639766],
        [ 0.26235187, -0.30335884, -0.87546644],
        [ 0.26613943, -0.39336442, -0.3230594 ],
        [-1.1633202 , -3.56296122,  4.24363177],
        [ 0.24785626, -0.10817446, -0.87789961],
        [ 0.02286978, -0.23829292, -0.49607114],
        [-0.14779748, -0.61277719, -0.07117618],
        [ 0.76918358, -0.5334122 , -0.50847697],
        [ 0.04484307, -0.20347597, -0.7231961 ],
```

```
       [-0.57859243, -3.56089679,  3.92529944],
       [ 0.3361223 , -0.09310822, -0.26152798],
       [ 0.42114203, -0.29484396, -0.82761875],
       [-0.11590082, -0.34413682, -0.5171558 ]])]
```

In [53]:
```
irisClf.coefs_
```

Out[53]:
```
[array([[-7.42716513e-02, -1.13514233e+00,  1.03188516e+00,
          1.21822319e-03, -7.12803793e-01,  6.06808089e-01,
         -6.21928282e-01,  5.71348293e-01, -7.24097095e-01,
         -8.47410629e-01, -7.30794358e-01, -1.76539842e-01,
         -5.38739601e-01,  3.69701099e-01, -1.03503796e-01],
        [-1.03521733e+00, -8.37645344e-01,  9.37905596e-01,
         -1.80960828e+00, -8.00864079e-01,  1.43947271e+00,
         -1.05139012e+00,  1.45984998e+00, -9.51563510e-01,
         -1.04508658e+00, -1.03416897e+00, -1.45053167e+00,
         -5.75247665e-01,  1.38547477e+00,  7.78199058e-01],
        [ 3.55493787e-01, -2.98735257e-01, -1.61321515e+00,
          1.66165360e+00, -5.16626276e-01, -1.45540278e+00,
          1.37173906e+00, -1.48759921e+00, -4.80140199e-01,
          1.46580991e+00, -3.42826524e-01,  1.64468358e+00,
          1.01840075e+00, -2.08008272e+00, -1.30068569e+00],
        [-3.18744898e-01,  1.55368347e-02, -2.70527422e+00,
          2.24821294e+00, -1.74521426e-01, -2.11584245e+00,
          1.75256523e+00, -1.99444460e+00, -5.24034802e-01,
          2.54351404e+00,  2.13776025e-02,  1.79119045e+00,
          1.96286382e+00, -1.95711640e+00, -1.42311398e+00]]),
 array([[ 7.37708281e-04,  1.36553268e+00, -5.28393611e-01],
        [-1.49652963e-01, -1.98410816e-01,  1.21010898e-01],
        [ 2.17939364e+00,  3.00892707e+00, -3.62779717e+00],
        [-2.00929025e+00,  1.18012887e+00,  5.64155904e-01],
        [-6.69425572e-01, -1.63010842e-01,  2.87575592e-01],
        [ 2.27675730e+00,  1.44566779e+00, -3.23368225e+00],
        [-2.59687588e+00, -1.64648038e-01,  3.00078897e+00],
        [ 2.11484816e+00,  9.36680875e-01, -3.03986470e+00],
        [-2.32943024e-01,  1.42753149e-01,  3.66500751e-01],
        [-1.10110165e+00, -1.21171525e+00,  2.27330992e+00],
        [ 3.07251337e-01, -1.02734510e-01, -4.92729469e-01],
        [-1.73079250e+00,  7.44767892e-01,  1.10760537e+00],
        [-1.01112585e+00, -9.31502200e-01,  1.39284301e+00],
        [ 3.27179712e+00, -3.06953658e+00, -1.84954861e+00],
        [ 4.26460244e-01,  1.87835319e-01, -2.49146579e-01]])]
```

In [54]:
```
len(irisClf.loss_curve_) # this tells me how many epochs it ran
```

Out[54]:
```
121
```

In [55]:
```
iris_predict_MLP = irisMLP.predict(values)
print(classification_report(iris_predict_MLP, labels))
print(confusion_matrix(iris_predict_MLP, labels))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.96      1.00      0.98        48
 Iris-virginica       1.00      0.96      0.98        52

       accuracy                           0.99       150
      macro avg       0.99      0.99      0.99       150
   weighted avg       0.99      0.99      0.99       150

[[50  0  0]
 [ 0 48  0]
 [ 0  2 50]]
```

In [56]:

```
iris_predict_clf = irisClf.predict(values)
print(classification_report(iris_predict_clf, labels))
print(confusion_matrix(iris_predict_clf, labels))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.96      0.98      0.97        49
 Iris-virginica       0.98      0.96      0.97        51

       accuracy                           0.98       150
      macro avg       0.98      0.98      0.98       150
   weighted avg       0.98      0.98      0.98       150

[[50  0  0]
 [ 0 48  1]
 [ 0  2 49]]
```

Their model did slightly worse than mine, despite the significantly better loss. Not sure why. I still trust theirs better because they have better engineers.

In [ ]: