Weights Top to bottom

= 4.80081929e-02   1.32088295e-01   3.10680451e-03   4.48595408e+00

= -4.69489522e-01   2.62621105e-01   2.40927137e-03   1.34846389e+01

= 7.76925263e-01  -2.57156596e-01   2.31112576e-03  -1.06534094e+01

= 1.84368277e-01  -1.13505466e-01   4.01820084e-03  -1.62881513e+01

= -4.85662465e-01  -2.87515419e-01   7.73294097e-01

= -4.31793064e-01   1.42782972e-01   2.87425731e-01

= -8.69078193e-03   1.11819396e-02  -1.13535433e-02

= 1.15467563e+01   8.36543598e+00  -1.99056196e+01

5.4053

-1.5276

-4.3231

9.3291

3.4927

-0.2448

-0.3976

**Training the Network:**

Based on 4 data points, sepal length, sepal width, petal length, and petal width I trained a neural network to classify different types of irises. To do that I preprocessed the data to normalize the data points so it was not the size of the data that mattered, but the ratio of sizes between the 4 data points to be able to more accurately identify the irises. The code can be seen below:

after the data was preprocessed I used scikitlearn’s neural network module to train a network to classify irises within the dataset I used LBFGS as the optimization method for my network, ReLU for the activation, and found the optimal hidden layer size to be 4.

*# Preprocess the data*

data\_new = []

**for** d **in** data:

data\_new.append(

*#np.array(d) / np.sum(d)*

np.array(d) / np.sqrt(np.sum(np.square(d)))

)

data = np.array(data\_new)

*# Train the neural network classifier.*

clf = MLPClassifier(

solver='lbfgs', *# lbfgs is good for small problems.*

activation='relu', *# options: logistic, tanh, relu, identity*

hidden\_layer\_sizes=(4), *# play with this!*

tol=1e-8 *# default: 1e-4*

)

To test if the network was able to generalize to irises that it wasn’t trained on I split the data set into two subsets, training and testing sets:

*#training data*

trData = np.array(data[:100])

trLabels = labels[:100]

*#testing data*

tData = data[100:]

tLabels = labels[100:]

The training set was composed of the first 100 data points in the set and the test set was composed of the last 50 data points. To ensure that the training set would not train on the first two types of irises and leave out the third I had to shuffle the data before assigning them to their subsets to ensure each iris was represented by the network

*#shuffle data so that irises of all types are in training set credit: https://stackoverflow.com/questions/43229034/randomly-shuffle-data-and-labels-from-different-files-in-the-same-order*

np.random.seed(3)

idx = np.random.permutation(len(data))

data,labels = data[idx], labels[idx]

np.random.seed()

the neural network was then trained on the training set and used again to see if it would generalize to the test set that I created. The results were pleasing on the test set I would sometimes receive a score higher than the training set for my last test I received 98% accuracy on the training set and 100% accuracy on the test set.