hw2_part2

October 16, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sympy as sy


import seaborn as sns
from sklearn.metrics import confusion_matrix

from IPython.display import display, Math, Latex

import nn
```

1 Part 2: Multinomial NN with Softmax, Categorical Cross Entropy, and One-Hot Encoding

1.1 Problem 7

1.1.1 Create Dataset

The data is meant to be RGB pairs, and the network has the task of sorting into red, green, blue, and black

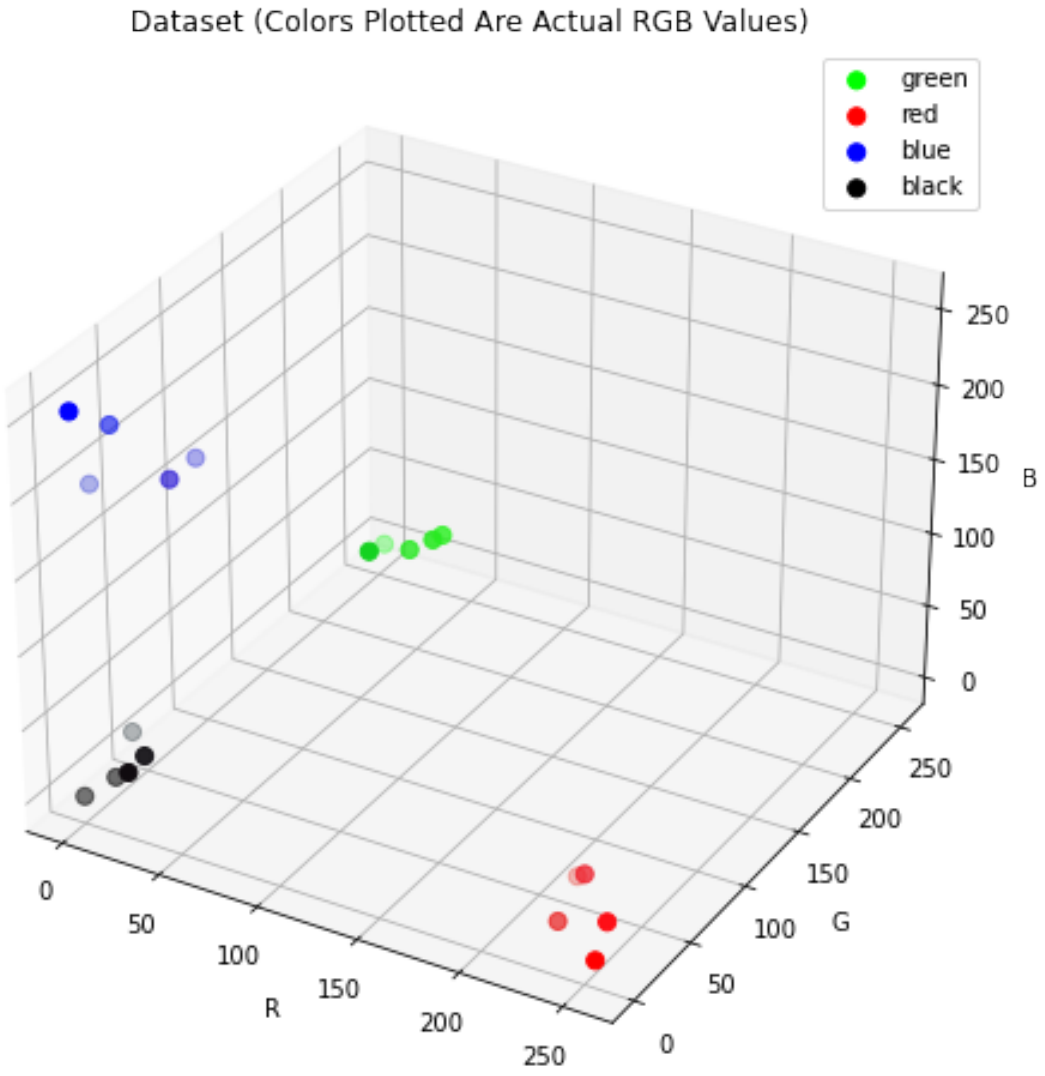
[ ]:
```
data = pd.read_csv("A2B_Data_EliWeissler.csv")
data
```

[ ]:

	R	G	B	LABEL
0	0	255	0	green
1	39	234	32	green
2	27	233	21	green
3	40	241	31	green
4	19	210	32	green
5	255	0	0	red
6	229	31	21	red
7	232	9	9	red
8	238	21	34	red
9	255	10	18	red
10	0	0	255	blue

```
11    8   18  238   blue
12   38   38  215   blue
13   37   19  213   blue
14    0   13  200   blue
15    0    0    0  black
16   10   10   10  black
17   20   18   23  black
18   10   24   31  black
19   16   11   15  black
```

[ ]:
```
fig = plt.figure()
ax = plt.axes(projection='3d')
fig.set_size_inches((8,8))
# Plotting
for i in ["green", "red", "blue", "black"]:
    data_subset = data[data["LABEL"] == i]
    ax.scatter(data_subset["R"], data_subset["G"], data_subset["B"],
               s=50, c=data_subset[["R", "G", "B"]].values/255, label=i)
ax.set_title('Dataset (Colors Plotted Are Actual RGB Values)')
ax.set_xlabel("R")
ax.set_ylabel("G")
ax.set_zlabel("B")
ax.legend()
plt.show()
```



1.1.2 Create X and Y

[ ]:
```
X, Y = nn.normalize_data(data, OHE=True)
Math("X = " + sy.latex(sy.Matrix(np.round(X, 2))) +\
     "\quad Y = " + sy.latex(sy.Matrix(Y)))
```

[ ]:

$$X = \begin{bmatrix} 0 & 1.0 & 0 \\ 0.15 & 0.92 & 0.13 \\ 0.11 & 0.91 & 0.08 \\ 0.16 & 0.95 & 0.12 \\ 0.07 & 0.82 & 0.13 \\ 1.0 & 0 & 0 \\ 0.9 & 0.12 & 0.08 \\ 0.91 & 0.04 & 0.04 \\ 0.93 & 0.08 & 0.13 \\ 1.0 & 0.04 & 0.07 \\ 0 & 0 & 1.0 \\ 0.03 & 0.07 & 0.93 \\ 0.15 & 0.15 & 0.84 \\ 0.15 & 0.07 & 0.84 \\ 0 & 0.05 & 0.78 \\ 0 & 0 & 0 \\ 0.04 & 0.04 & 0.04 \\ 0.08 & 0.07 & 0.09 \\ 0.04 & 0.09 & 0.12 \\ 0.06 & 0.04 & 0.06 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.1.3 Build the network and verify that we are handling the softmax Jacobian correctly:

[ ]:
```
# Initialize network
input_size = 3
hidden_layers = [2]
output_size = 4
activation_fns = [nn.ReLU, nn.softmax]
loss_fn = nn.loss_CCE
random_initialize = True
network = nn.NeuralNetwork(input_size, output_size, hidden_layers,
                           activation_fns, loss_fn,␣
 ↪random_initialize=random_initialize)
```

[ ]:
```
# Do the machinations needed to get dL/dz2
X_subset = X[0]
Y_subset = Y[0]
pred = network.feed_forward(X_subset)
network.calc_jacobian()
network.back_propagate(X_subset, Y_subset, pred, lr=1)
```

[ ]:
```
network.dLdz[1]
```

[ ]:
```
array([-0.8324437 ,  0.18753036,  0.44529529,  0.19961804])
```

[ ]:
```
pred-Y_subset
```

[ ]:
```
array([[-0.8324437 ,  0.18753036,  0.44529529,  0.19961804]])
```

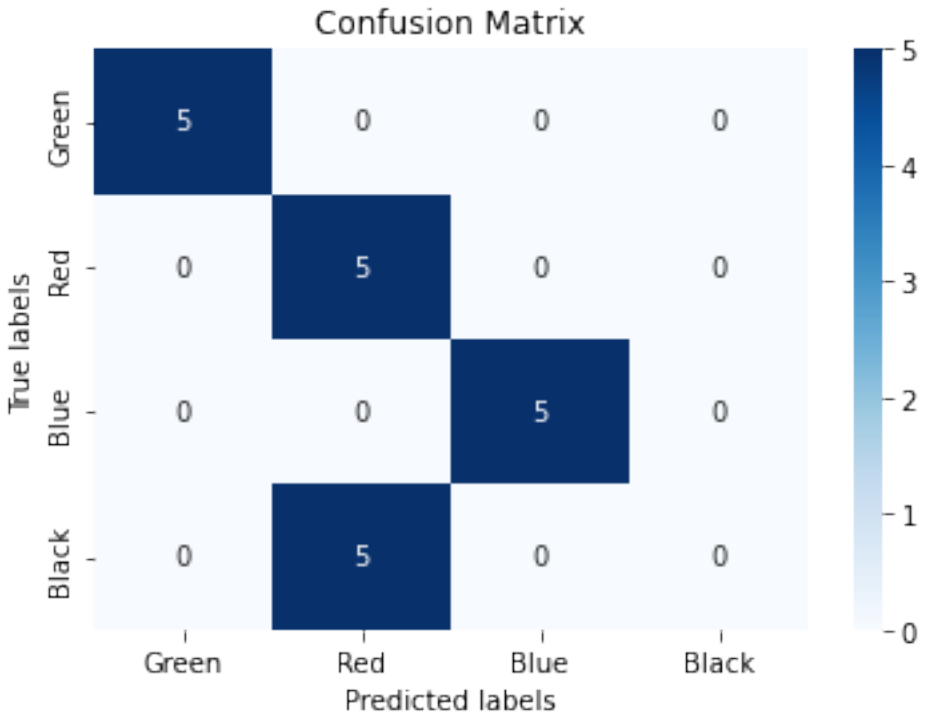
Yay it's calculating it correctly!

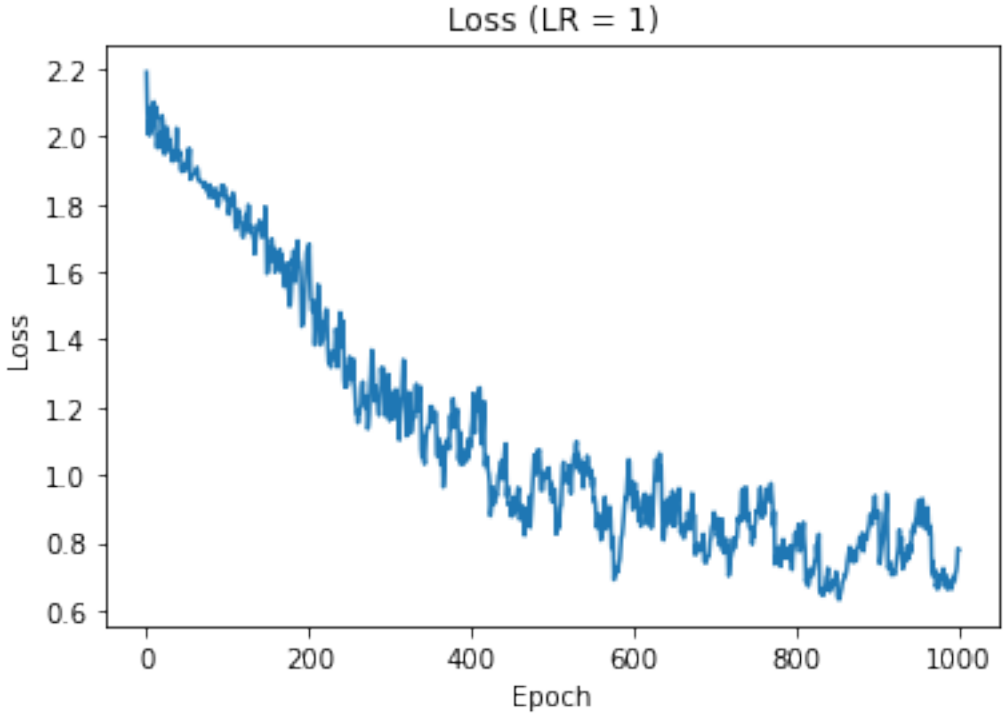
1.1.4 Now try actually training:

[ ]:
```
# Initialize network
input_size = 3
hidden_layers = [2]
output_size = 4
activation_fns = [nn.ReLU, nn.softmax]
loss_fn = nn.loss_CCE
random_initialize = True
network = nn.NeuralNetwork(input_size, output_size, hidden_layers,
                           activation_fns, loss_fn,␣
 ↪random_initialize=random_initialize)
# Train network
epochs = 1000
lr = 1
batch_size = 9
loss = network.train(X, Y, X, Y, epochs=epochs, lr=lr, batch_size=batch_size,␣
 ↪check_progress=150)
```

```
Epoch 0 (out of 1000) -- Loss: 2.1901
Epoch 150 (out of 1000) -- Loss: 1.6439
Epoch 300 (out of 1000) -- Loss: 1.2229
Epoch 450 (out of 1000) -- Loss: 0.9065
Epoch 600 (out of 1000) -- Loss: 0.935
Epoch 750 (out of 1000) -- Loss: 0.8934
Epoch 900 (out of 1000) -- Loss: 0.8929
```

[ ]:
```
# Predict and plot
pred = network.feed_forward(X)
nn.plot_confusion_matrix(Y, pred, labels=["Green", "Red", "Blue", "Black"])
nn.plot_loss(loss, lr)
```


Confusion Matrix
Green
Red
Blue
Black
True labels
5 0 0 0
0 5 0 0
0 0 5 0
0 5 0 0
Green
Red
Blue
Black
Predicted labels
5
4
3
2
1
0

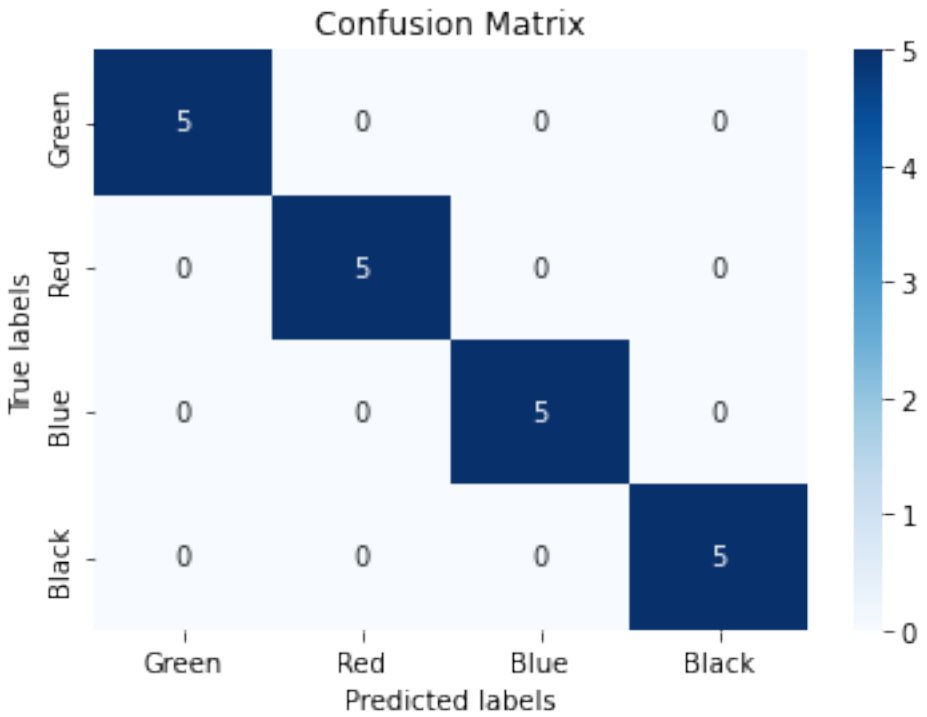

Loss (LR = 1)
Loss
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0
200
400
600
800
1000
Epoch

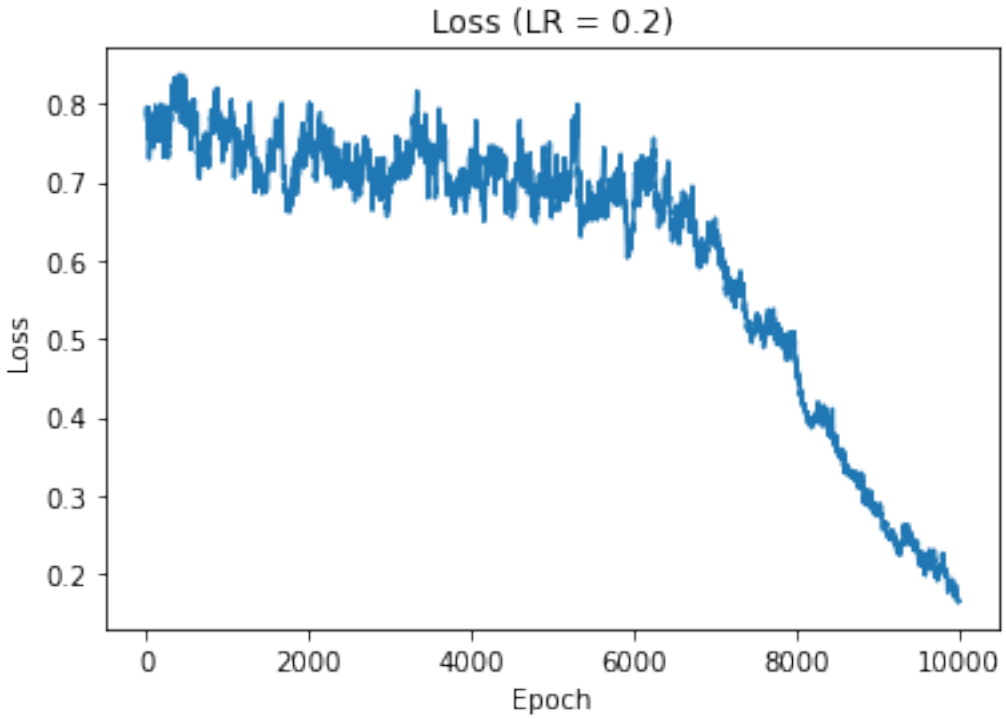
We're correctly predicting three of the colors after 1000 epochs, but we are missing black. So try training longer to see if we can get past that. This will continue where the last training left off:

[ ]:
```
# Train network
epochs = 10000
lr = 0.2
batch_size = 9
loss = network.train(X, Y, X, Y, epochs=epochs, lr=lr, batch_size=batch_size,␣
 ↪check_progress=1000)
```

```
Epoch 0 (out of 10000) -- Loss: 0.7807
Epoch 1000 (out of 10000) -- Loss: 0.7617
Epoch 2000 (out of 10000) -- Loss: 0.7643
Epoch 3000 (out of 10000) -- Loss: 0.6989
Epoch 4000 (out of 10000) -- Loss: 0.7403
Epoch 5000 (out of 10000) -- Loss: 0.6674
Epoch 6000 (out of 10000) -- Loss: 0.6696
Epoch 7000 (out of 10000) -- Loss: 0.654
Epoch 8000 (out of 10000) -- Loss: 0.4533
Epoch 9000 (out of 10000) -- Loss: 0.2773
```

[ ]:
```
# Predict and plot
pred = network.feed_forward(X)
nn.plot_confusion_matrix(Y, pred, labels=["Green", "Red", "Blue", "Black"])
nn.plot_loss(loss, lr)
```





Yay we got off of the plataeu! Instead of training the network longer, we could have also restarted with a different random initial guess for weights/biases, and that may have also gotten us to where we wanted to go!