CS390-NIP - Fall 2020
Name: Michael Lee
Email: lee2965@purdue.edu
Date: September 21, 2020

Lab 1 Report

Basic Image Classification

Due Date: September 18, 2020

https://github.com/michaeldlee23/cs390-lab1

Resources used

- Explanation of calculus
  https://www.youtube.com/watch?v=GlcnxUlrtek&t=390s
- Starting point for determining number of hidden neurons to use
  https://stats.stackexchange.com/a/136542
- Tensorflow and Keras model creation, training, predicting, optimizing
  https://www.tensorflow.org/api_docs/python/tf/keras/

Completed portions of lab

1. [30] Custom Neural Net
   - ✓[5] Sigmoid and sigmoid derivative functions
   - ✓[10] Train Function (using backpropagation properly)
   - ✓[15] Fully functioning 2-layer neural net
   - ✓[+3] EC: implement an option to make the network 3 layer
   - ✓[+5] EC: make the network N layer, where N is given in the constructor
   - ✓?[+1] EC: implement a second activation function (ReLU) that your network can use. The activation should be specified in the constructor
2. [30] TF Neural Net
   - ✓[30] Implement a function that builds a 2 layer neural net using either Tensorflow or Keras.
   - [+6] EC: Get accuracy to 99%.
3. [20] Pipeline & Misc.
   - ✓[5] Preprocess data such that values are between 0.0 and 1.0
   - ✓[15] Create an F1 score confusion matrix in the evaluation function and print it
   - ✓[+5] EC: Look up and learn about the iris dataset. Download the dataset and write a program to classify it using your custom neural net.
4. ✓[20] Report

Summary

- Custom Neural Network

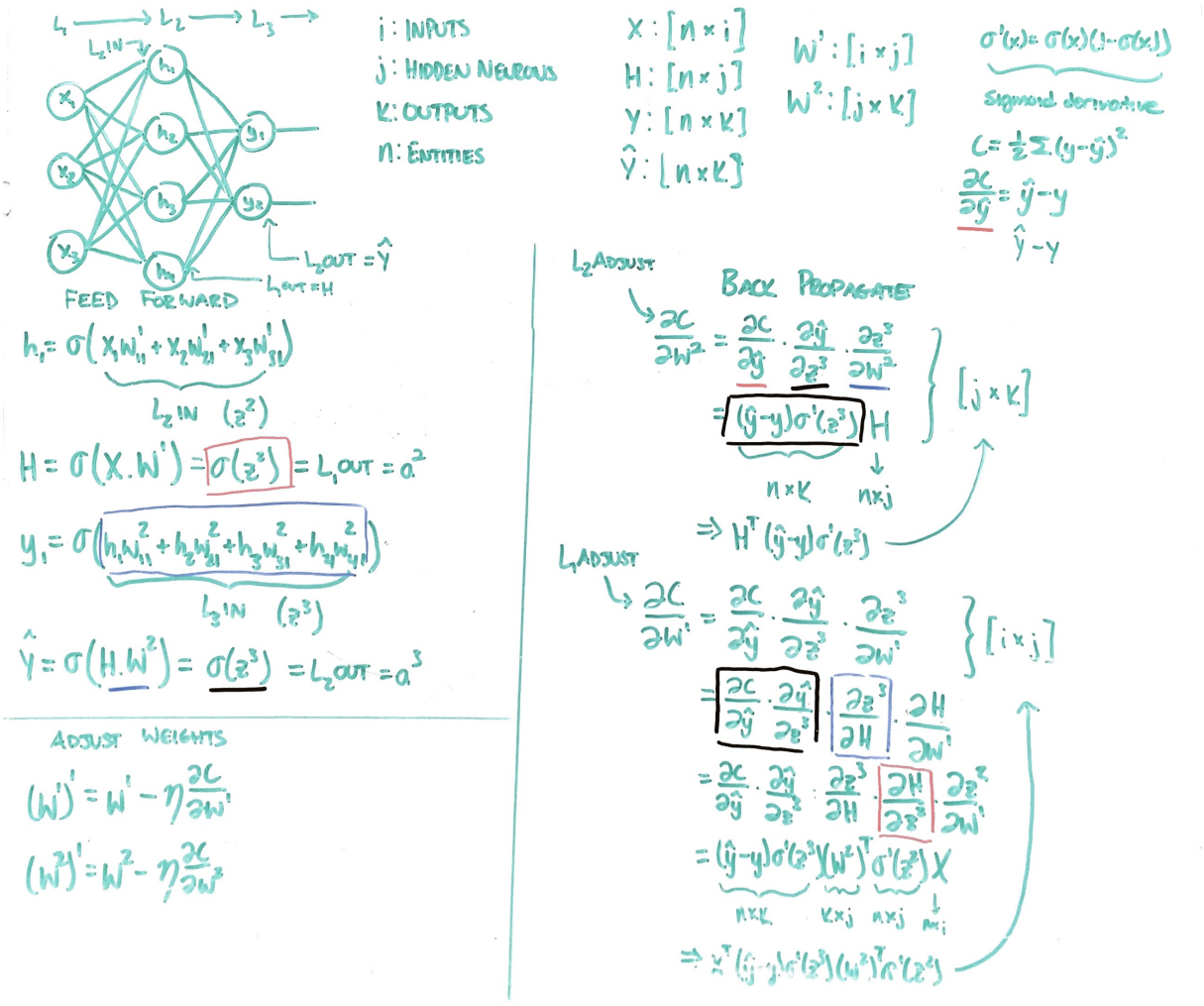


Figure 1: Work through backpropagation calculus

In order to understand how to implement the backpropagation algorithm, I needed to fully understand the math behind the process. I heavily relied on my whiteboarded work and constantly referenced it while I implemented the custom neural net, as it proved to be extremely helpful in keeping track of the dimensions of inputs. It also helped me keep track of variable names, as there were so many different values and matrices to keep track of.

In order to make testing easier, I implemented command line arguments to adjust hyperparameters:

```
Usage:
-a [algorithm | guesser, custom_net, tf_net]
-e [number of epochs]
-b [batch size]
-n [number of hidden neurons]
-l [initial learning rate]
-f [activation function | sigmoid, relu]
```

- Tensorflow Neural Network
  The TensorFlow network was implemented in a similar manner that was shown on the lecture slides. Playing around with the hyperparamters (number of epochs and number of hidden neurons), we can see that the TensorFlow network performs significantly better than the custom net, as expected. In attempts to achieve 99% accuracy, I added layers, increased the number of hidden neurons, and tried different optimizers and batch sizes. Accuracy persistently hovered at around 98.2%.

Relevant outputs

1. F1 Confusion Matrix

```
HYPERPARAMETERS:
net: custom_net epochs: 500     batchSize: 100  neuronsPerLayer: 5      learningRate: 0.1

Classifier algorithm: custom_net
Classifier accuracy: 57.640000%
Classifier confusion matrix:
[[  931.     2.    73.    73.     7.   646.    12.   770.   736.    40.  3290.]
 [    0.  1086.    22.    23.     3.     7.     1.    17.    12.     1.  1172.]
 [   17.    20.   882.    23.     3.    22.    41.    30.    43.     2.  1083.]
 [    2.    15.     7.   853.     4.    66.     0.    23.    37.    14.  1021.]
 [    7.     2.    11.     3.   892.    26.    30.    26.    21.   772.  1790.]
 [    2.     1.     2.    11.     0.    29.     0.     2.     5.     2.    54.]
 [    3.     6.    25.     0.    26.     6.   844.     2.    14.     5.   931.]
 [    9.     2.     0.    13.    11.    56.    17.    86.    60.    31.   285.]
 [    2.     1.     8.     8.     0.    10.     0.    14.    19.     0.    62.]
 [    7.     0.     2.     3.    36.    24.    13.    58.    27.   142.   312.]
 [  980.  1135.  1032.  1010.   982.   892.   958.  1028.   974.  1009. 10000.]]

Classifier F1 score matrix:
[[0.4361    nan    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan 0.9415    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan 0.834     nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan 0.84      nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan 0.6436    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan 0.0613    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan 0.8936    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan 0.131     nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan 0.0367    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan    nan 0.215 ]]
```

Epochs: 500, Neurons/Layer: 5

```
HYPERPARAMETERS:
net: custom_net epochs: 500     batchSize: 100  neuronsPerLayer: 10     learningRate: 0.1

Classifier algorithm: custom_net
Classifier accuracy: 90.100000%
Classifier confusion matrix:
[[  909.     0.     8.    20.     0.    24.     8.     1.     4.     6.   980.]
 [    0.  1100.     2.     0.     0.     4.     3.    11.     4.     4.  1128.]
 [    7.     9.   927.    21.     5.     8.    10.    28.     8.     1.  1024.]
 [   11.     0.    27.   871.     0.    38.     9.     7.    21.    21.  1005.]
 [    1.     1.     9.     3.   915.     5.     9.    17.     8.    34.  1002.]
 [   31.     1.     1.    26.     4.   730.    14.     0.    50.     7.   864.]
 [   14.     5.    17.     7.    23.    29.   898.     3.    15.     3.  1014.]
 [    5.     2.    15.    13.     1.     5.     1.   920.     4.    19.   985.]
 [    1.    16.    22.    43.     4.    37.     6.     4.   847.    21.  1001.]
 [    1.     1.     4.     6.    30.    12.     0.    37.    13.   893.   997.]
 [  980.  1135.  1032.  1010.   982.   892.   958.  1028.   974.  1009. 10000.]]

Classifier F1 score matrix:
[[0.9276    nan    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan 0.9722    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan 0.9018    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan 0.8645    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan 0.9224    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan 0.8314    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan 0.9108    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan 0.9141    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan 0.8577    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan    nan 0.8903]]
```

Epochs: 500, Neurons/Layer: 10

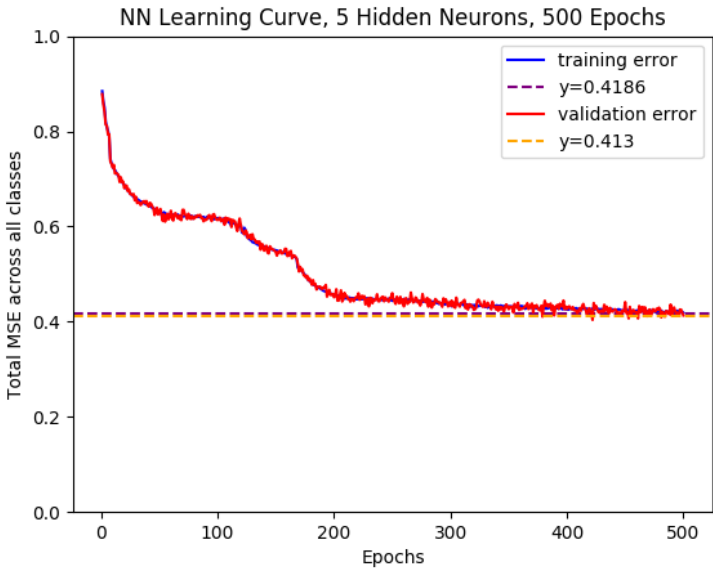
```
HYPERPARAMETERS:
net: custom_net epochs: 500     batchSize: 100  neuronsPerLayer: 20     learningRate: 0.1

Classifier algorithm: custom_net
Classifier accuracy: 92.120000%
Classifier confusion matrix:
[[  952.     0.     6.     2.     2.    11.    13.     1.     3.     9.   999.]
 [    0.  1115.     1.     1.     2.     3.     3.    11.     5.     4.  1145.]
 [    3.     0.   923.    27.     7.     4.     9.    22.     8.     1.  1004.]
 [    0.     7.    14.   914.     0.    32.     1.    13.    31.    21.  1033.]
 [    1.     0.    12.     0.   922.     5.     9.     5.    12.    38.  1004.]
 [    4.     0.     8.    29.     3.   786.    18.     0.    21.     8.   877.]
 [   12.     3.    18.     5.     9.    17.   899.     3.    12.     1.   979.]
 [    3.     1.    13.    14.     1.    12.     1.   949.    10.    24.  1028.]
 [    5.     9.    31.    13.     4.    19.     5.     3.   861.    12.   962.]
 [    0.     0.     6.     5.    32.     3.     0.    21.    11.   891.   969.]
 [  980.  1135.  1032.  1010.   982.   892.   958.  1028.   974.  1009. 10000.]]

Classifier F1 score matrix:
[[0.9621    nan    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan 0.9781    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan 0.9067    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan 0.8948    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan 0.9285    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan 0.8886    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan 0.9282    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan 0.9232    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan 0.8895    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan    nan 0.9009]]
```

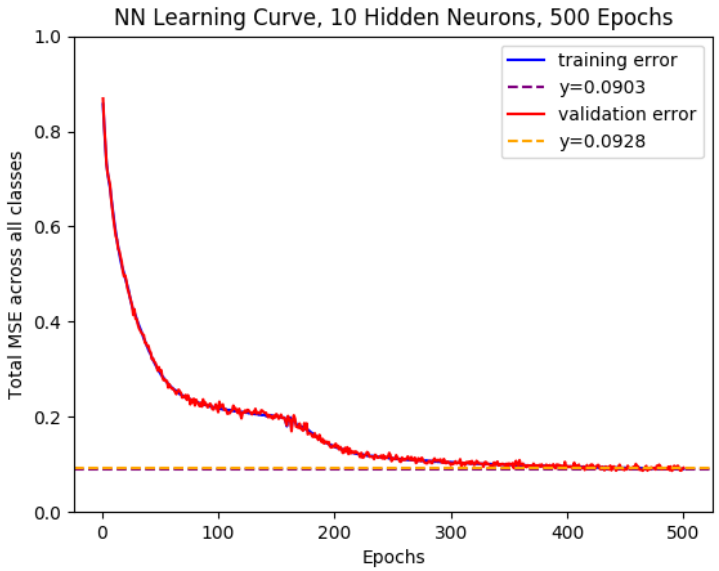
Epochs: 500, Neurons/Layer: 20

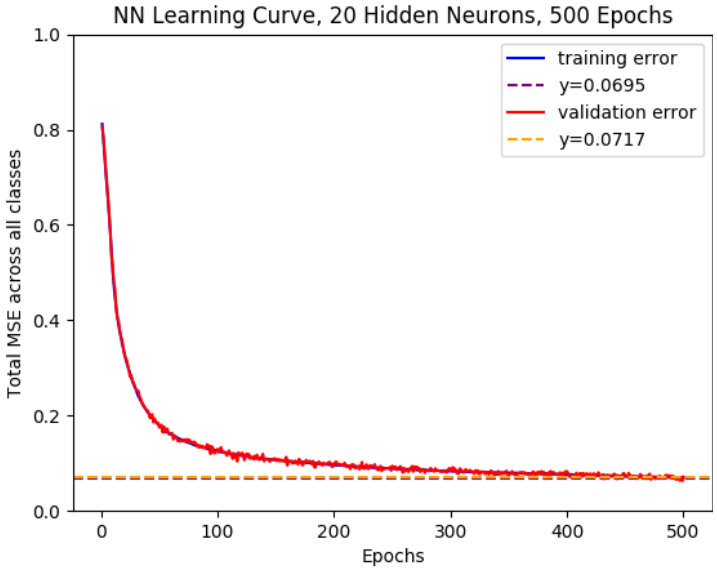
```
HYPERPARAMETERS:
net: custom_net epochs: 1000    batchSize: 100  neuronsPerLayer: 20     learningRate: 0.1

Classifier algorithm: custom_net
Classifier accuracy: 93.140000%
Classifier confusion matrix:
[[  951.     0.    10.     2.     2.     7.    15.     2.     3.     9.  1001.]
 [    0.  1119.     5.     2.     0.     3.     3.     8.     4.     7.  1151.]
 [    3.     1.   936.    28.     7.     3.     6.    21.     8.     0.  1013.]
 [    0.     6.    12.   921.     0.    28.     1.    10.    22.    19.  1019.]
 [    2.     0.    10.     1.   921.     2.     6.     3.    10.    34.   989.]
 [    3.     0.     6.    20.     2.   804.    14.     0.    14.     7.   870.]
 [   12.     2.    14.     6.    13.    16.   906.     2.    10.     2.   983.]
 [    3.     1.    12.    13.     2.     6.     2.   966.     8.    19.  1032.]
 [    5.     6.    22.    14.     3.    20.     5.     2.   889.    11.   977.]
 [    1.     0.     5.     3.    32.     3.     0.    14.     6.   901.   965.]
 [  980.  1135.  1032.  1010.   982.   892.   958.  1028.   974.  1009. 10000.]]

Classifier F1 score matrix:
[[0.9601    nan    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan 0.979     nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan 0.9154    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan 0.9078    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan 0.9346    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan 0.9126    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan 0.9335    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan 0.9379    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan 0.9113    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan    nan 0.9129]]
```

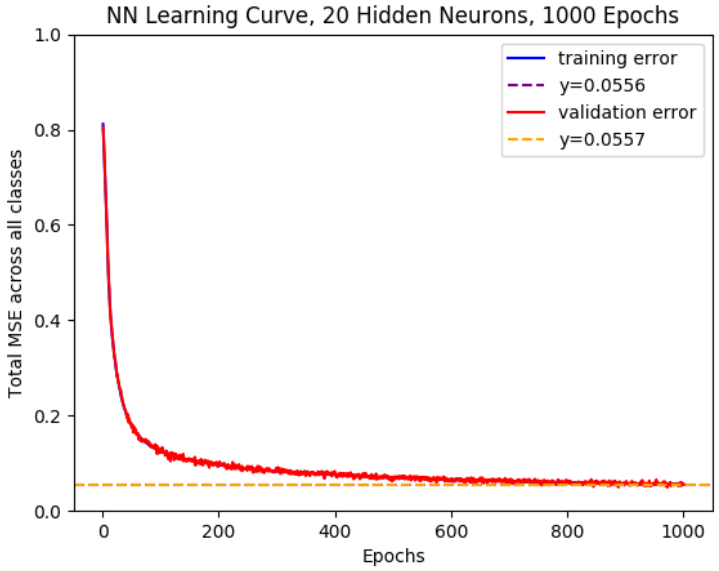
Epochs: 1000, Neurons/Layer: 20

2. Error Plots







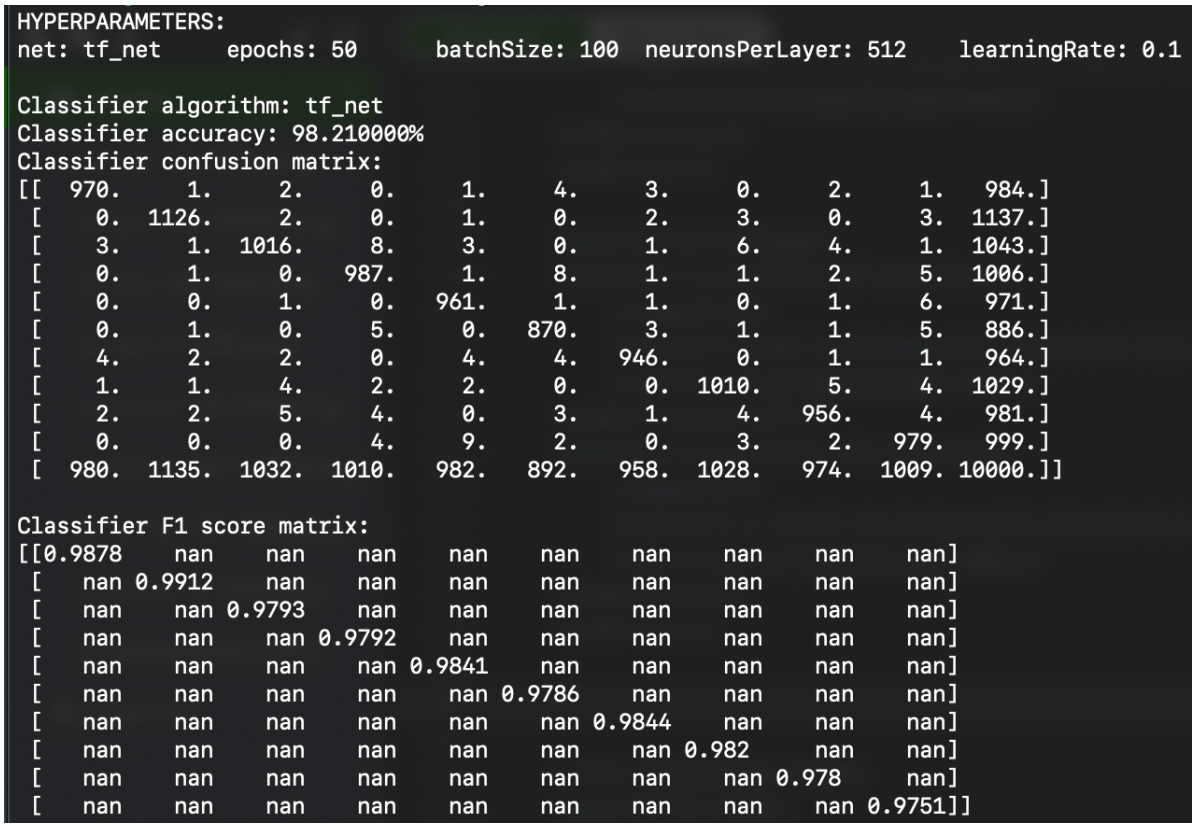


To evaluate my model's learning curve, I used cross validation with a 90-10 split. At the beginning of each epoch, I randomly shuffle the inputted data sets and partition 10% to be used for validation, while the rest is then used to train the model. At the end of the epoch, I run the model against the training and validation set and calculate the total mean square error across all output classes. As we can see from the outputted graphs, the model appears to not overfit at all, as the training and validation error over time remains practically identical.

To observe the effects of certain hyperparameters (namely epoch number and hidden

layer size), I tested the network under several different cases. We can see that increasing the hidden layer size drastically increases accuracy from 5 to 10, but its effects diminish as we go from 10 to 20.

3. TensorFlow Output



Epochs: 50, Neurons/Layer: 512

4. Extra Content

- N-Layer Custom Net
  I implemented an N-Layer custom net in `extra.py`. I also implemented command-line argument parsing to easily set hyperparameters; using the `-l` flag will set the number of hidden layers. Below is the output for a network with 3 hidden layers, each layer having 10 neurons, trained over 500 epochs.

```
HYPERPARAMETERS:
net: custom_net epochs: 500     batchSize: 100  hiddenSize: 10  hiddenLayers: 3 activation: sigmoid

Classifier algorithm: custom_net
Classifier accuracy: 88.470000%
Classifier confusion matrix:
[[  922.     0.    16.    18.     0.    11.    15.     9.     3.     6.  1000.]
 [    0.  1095.    21.     5.     2.     2.     2.    16.     9.     1.  1153.]
 [    9.     9.   877.    29.     2.    11.    14.    39.    20.     4.  1014.]
 [    4.     5.    21.   858.     0.    55.     1.    11.    12.    12.   979.]
 [    7.     0.     7.     1.   881.    11.    20.     9.     7.    57.  1000.]
 [    7.     3.    10.    55.     1.   731.     7.     2.    68.    17.   901.]
 [   17.     5.    23.     3.    24.    17.   891.     0.    12.     0.   992.]
 [    9.     3.    10.     8.     5.    11.     2.   898.    12.    16.   974.]
 [    5.    14.    41.    24.    15.    37.     6.    13.   814.    16.   985.]
 [    0.     1.     6.     9.    52.     6.     0.    31.    17.   880.  1002.]
 [  980.  1135.  1032.  1010.   982.   892.   958.  1028.   974.  1009. 10000.]]

Classifier F1 score matrix:
[[0.9313    nan    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan 0.9572    nan    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan 0.8573    nan    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan 0.8627    nan    nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan 0.889     nan    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan 0.8154    nan    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan 0.9138    nan    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan 0.8971    nan    nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan 0.831     nan]
 [   nan    nan    nan    nan    nan    nan    nan    nan    nan 0.8752]]
```

Epochs: 500, Neurons/Layer: 10, Hidden Layers: 3

- ReLU Activation
  I attempted to implement the option to use a ReLU activation function instead of sigmoid, however there are some unexpected results. I have implemented the `__relu()` and `__reluDerivative()` functions that appear to behave as intended, but when actually running the model using these activation functions we get very poor accuracy — the model tends to only pick 0s and 1s. I tried using a smaller learning rate (0.001 instead of 0.1) which had minimal improvement. I also tried using ReLU on just the hidden layer, and kept the output layer as sigmoid, but that also had minimal improvement.
- Iris Dataset
  The option to run the neural net on the iris dataset was implemented in `extra.py`, which can be specified using the `-d iris` flag. Here are the results of running the

custom net on the iris dataset with 1 hidden layer, 20 hidden neurons, and 1000 epochs:

```
HYPERPARAMETERS:
net: custom_net epochs: 1000    batchSize: 100  hiddenSize: 20  hiddenLayers: 1

Classifier algorithm: custom_net
Classifier accuracy: 93.333333%
Classifier confusion matrix:
[[15.  0.  0. 15.]
 [ 0. 13.  2. 15.]
 [ 0.  1. 14. 15.]
 [15. 14. 16. 45.]]

Classifier F1 score matrix:
[[1.        nan    nan]
 [   nan 0.8966    nan]
 [   nan    nan 0.9032]]
```

Epochs: 1000, Neurons/Layer: 20, Hidden Layers: 1

The iris dataset was obtained through python's `sklearn.datasets` package, and contains 150 training samples. Since no test set is provided, I had to shuffle the given data and arbitrarily took 45 samples to set aside as the test set. The remaining 105 sample were used to train the model.