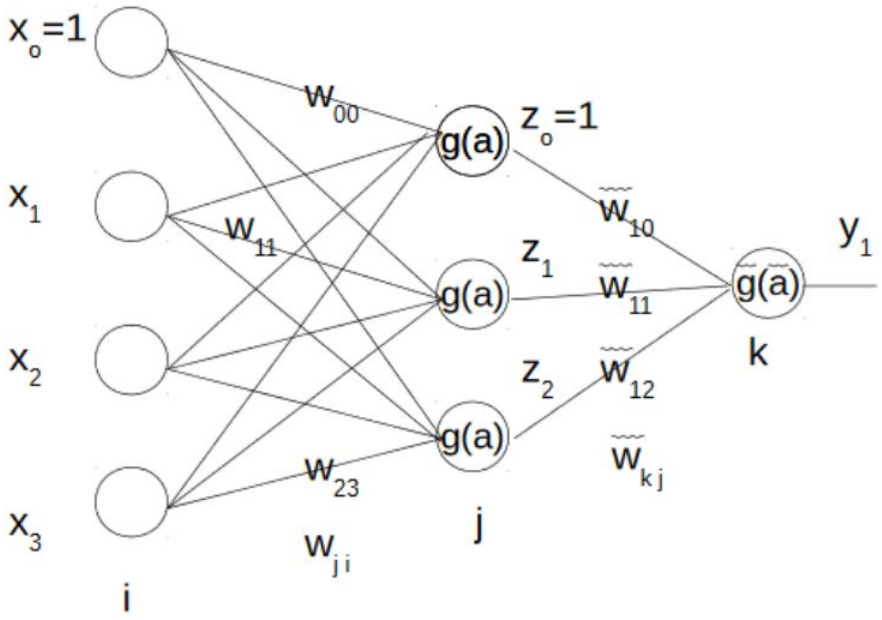
2019

Applied Physics 186
Activity 17

Neural Networks

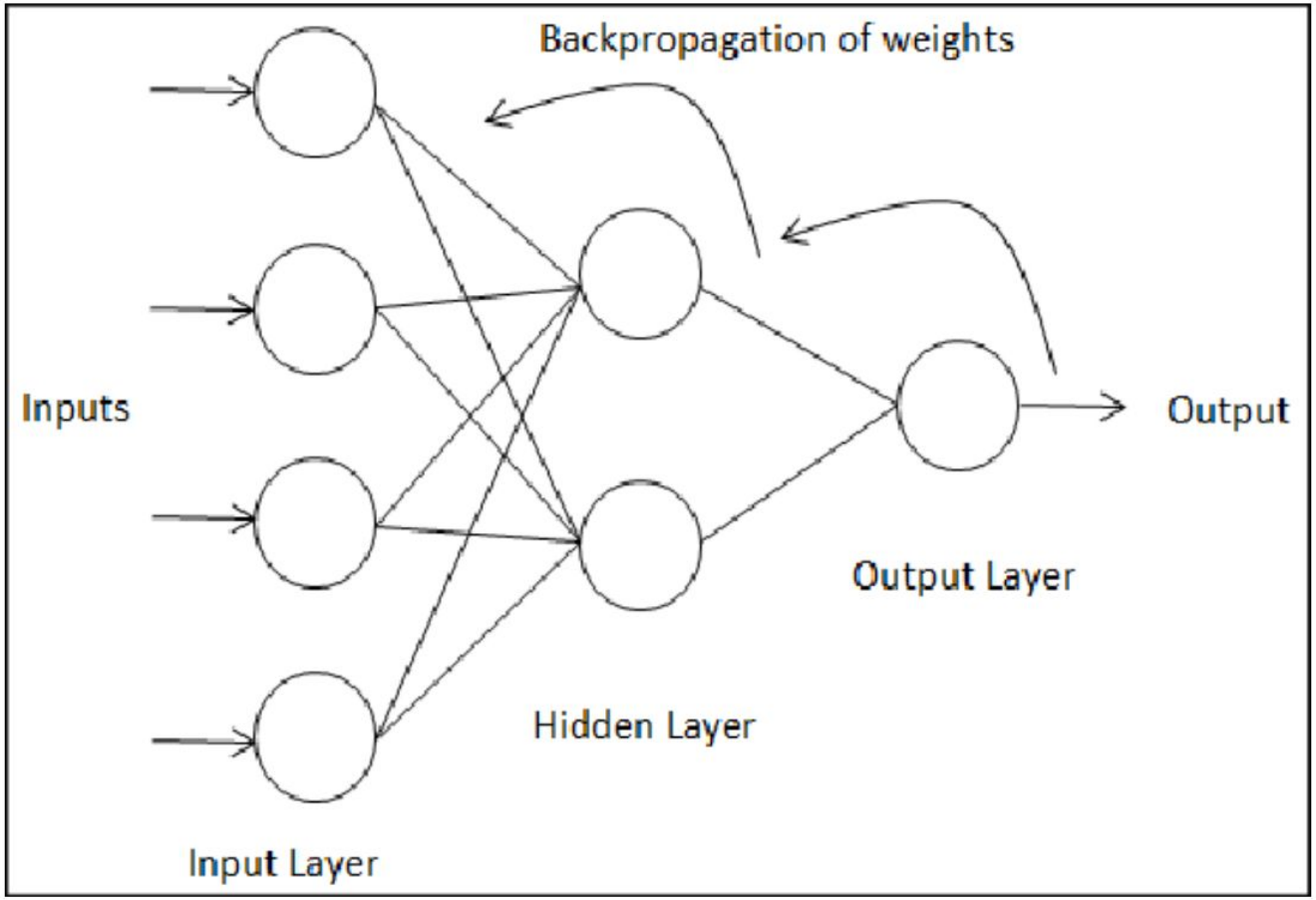
Castro, Marc Jerrone R.
B.S. Applied Physics
2015-07420

Neural Networks



Common Use & Purpose: Neural Networks are multi-layered perceptrons that utilize several layers containing at least 1 input layer, one or more hidden layers, and finally an output layer.

Neural Networks is a type of deep learning algorithm that uses basic concepts of machine learning that is known for its ability to perform error back-propagation

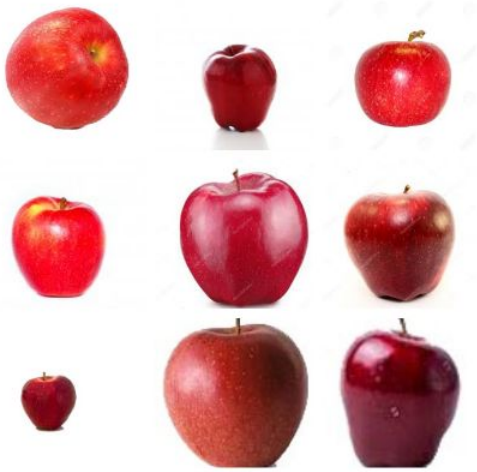


Neural Networks
Back-propagation

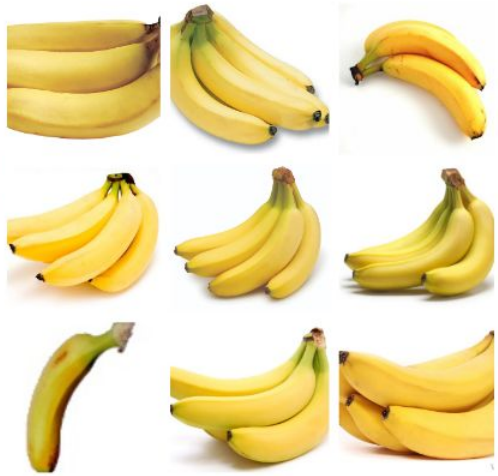
Objective: To calculate the gradient descent for weight updates for each feature input. Gradients for the output layer is calculated first, with the direction of determination going from deeper layers to shallow layers such that for the final the first layer's gradient of weights are calculated last.

REFERENCE: https://deepai.org/machine-learning-glossary-and-terms/backpropagation

Data and Inputs



The data used for this study is a set of images of fruits (apples, bananas, oranges) taken from Google Images and portion of the Fruits360 dataset.



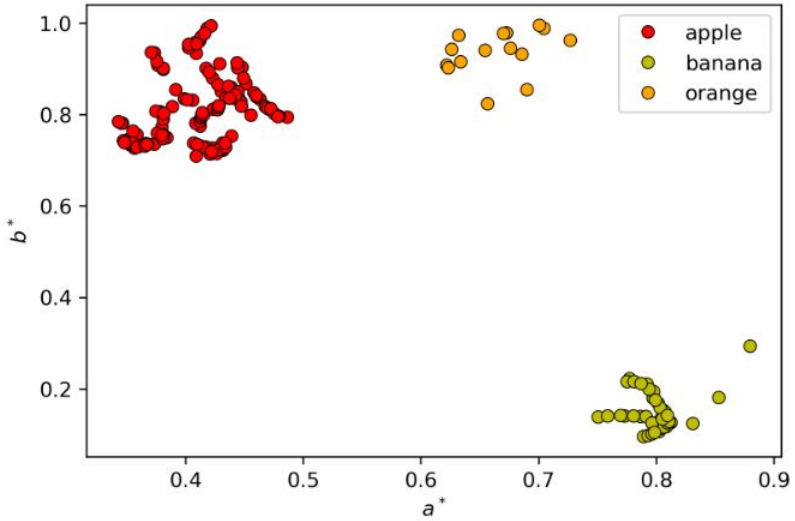
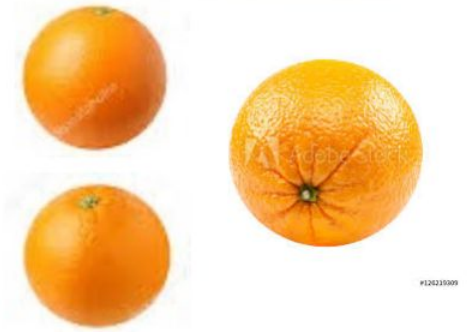


Figure 1. Feature space of banana and apple properties. This particular figure presents the comparison of the average pixel value and circularity for each sample image.



Using the same pre-processing steps from Activity 11 - Feature Extraction, image features such as average pixel value and circularity were extracted from each image.

Neural Networks for Curve-Fitting

In this activity, we attempted to explore the application of a neural network using a multi-layer perceptron to fit a variety of functions such as:

A sine wave function

```
y_train = np.sin(np.pi*X_train)
```

A quadratic function

```
y_train = X_train**2
```

A cubic function

```
y_train = X_train**3
```

A hyperbolic tangent function

```
y_train = np.tanh(np.pi*X_train)
```

A sigmoid function

```
y_train = 1/(1+ np.exp(-1*X_train*np.pi))
```

Each of these functions were generated and configured as to achieve optimal characteristics for our objective of fitting functions.

X_train can be attributed to the following function:

```
np.random.seed(186)
X_train = rand.uniform(-1, 1, 500)
```

Whereas, we generated 500 random values - with the insurance that the same value are randomized for each run by loading a specific random seed.

We then performed data augmentation by generating powers of each value from X_train using:

```
1  powers = 5
2  ffs = []
3  for i in X_train:
4    exponent = 0
5    feature_space = []
6    while exponent <= powers:
7      feature_space.append(i**exponent)
8      exponent+=1
9    ffs.append(feature_space)
```

We got this idea from the concept that any function can be described as :

$$f(x) = w_o + w_1 x + w_2 x^2 + w_3 x^3 + \dots \quad .$$

where x are values of each member from X_train and w are coefficients or weights for each of the powers.

Neural Networks for Curve-Fitting

We then insert these powers as a feature space that corresponds to our X_train values with y values equal to their corresponding function values:

	X0	X	X2	X3	X4	X5	Y
0	1.0	-0.900034	0.810062	-0.729083	0.656200	-0.590602	0.587611
1	1.0	-0.896720	0.804108	-0.721060	0.646589	-0.579810	0.604330
2	1.0	-0.860331	0.740169	-0.636790	0.547850	-0.471332	0.769187
3	1.0	-0.846841	0.717140	-0.607304	0.514290	-0.435522	0.820522
4	1.0	-0.800847	0.641356	-0.513628	0.411337	-0.329418	0.949399

We then inserted these functions into our neural network using the following function:

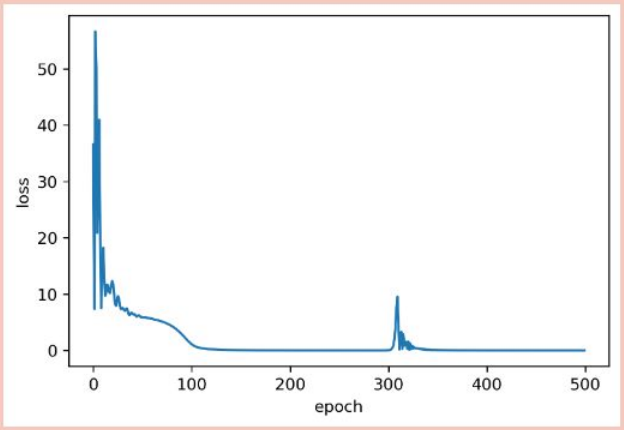
Whereas, each of the values of X were paired with a corresponding randomized weight and fed forward into our neural network. We utilized the rectified linear unit activation function to transfer the input data into our hidden layer whilst calculating our loss for each iteration.

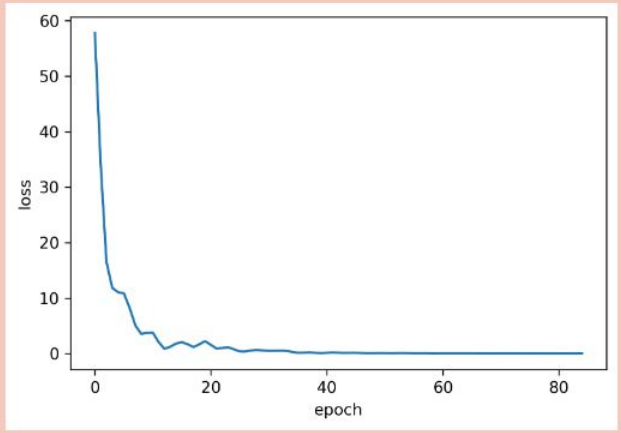
I set conditional statements such that it would stop the run once the generated loss is approximately 0.01 via this answer I found on Stack Overflow (https://stackoverflow.com/a/37296168).

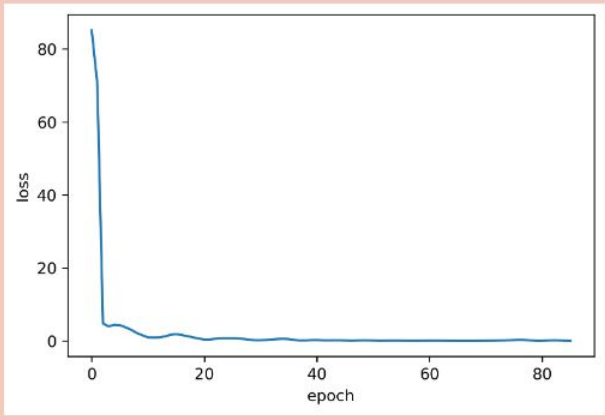
```
def nInit(self):
    np.random.seed(1)
    self.param['W1'] = np.random.randn(self.dims[1], self.dims[0]) / np.sqrt(self.dims[0])
    self.param['b1'] = np.zeros((self.dims[1], 1))
    self.param['W2'] = np.random.randn(self.dims[2], self.dims[1]) / np.sqrt(self.dims[1])
    self.param['b2'] = np.zeros((self.dims[2], 1))
    return

def forward(self):
    Z1 = self.param['W1'].dot(self.X) + self.param['b1']
    A1 = Relu(Z1)
    self.ch['Z1'],self.ch['A1']=Z1,A1

    Z2 = self.param['W2'].dot(A1) + self.param['b2']
    A2 = Linear(Z2)
    self.ch['Z2'],self.ch['A2']=Z2,A2

    self.Yh=A2
    loss=self.nloss(A2)
    return self.Yh, loss

def nloss(self,Yh):
    loss = (1./self.sam) * (-np.dot(self.Y,np.log(Yh).T) - np.dot(1-self.Y, np.log(1-Yh).T))
    return loss
```

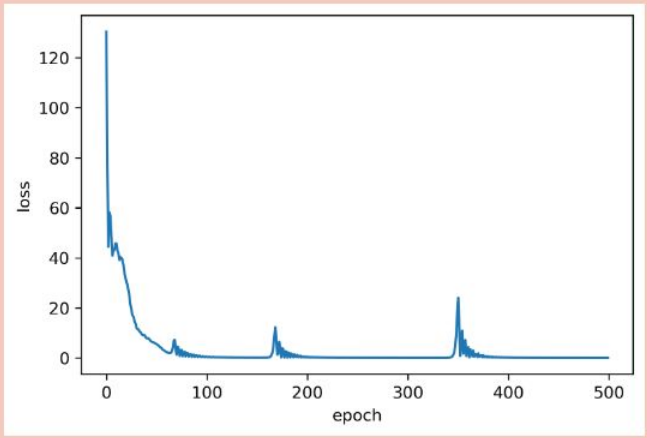
The loss at each iterattion was then recorded and plotted. The results for each function are as follows:

Neural Networks for Curve-Fitting









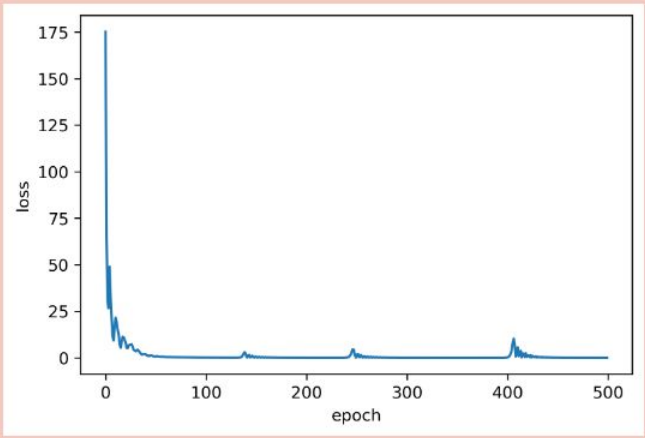


Figure 2. Loss at each epoch for varying functions. (Left to right) Cubic, Quadratic, Sigmoid, Sine function, hyperbolic tangent.

Neural Networks for Curve-Fitting

As observed from each of the graphs from the previous slide, The loss at each epoch approaches zero loss for increasing epochs. However, one may notice significant peaks located in between the initial and final epoch. This suggests that the derived model deviated from the expected results, however if one may notice, by increasing the number of epochs, the quality or the characteristics of the plot are generalized such that it depicts an erroneous value as the initial loss as slowly approached a plateau which suggests it had reached the optimal value of loss = 0.01

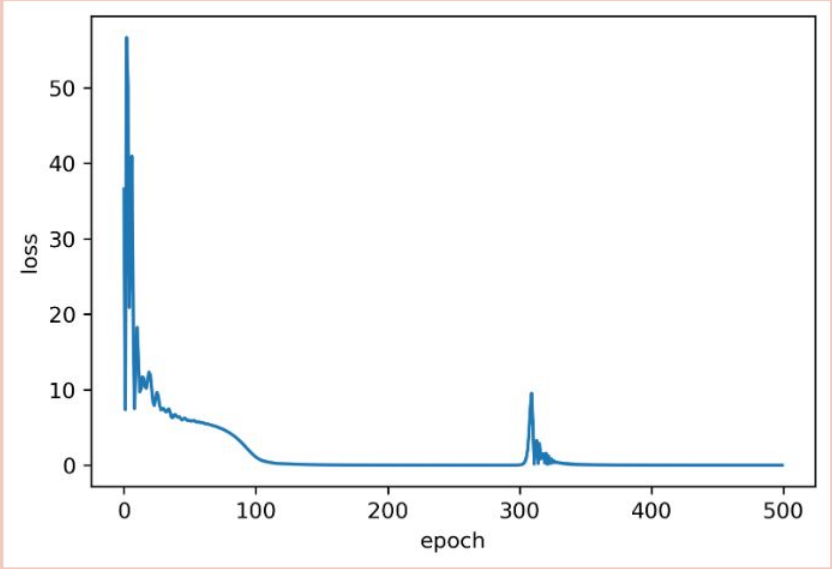


Figure 3. Loss at each epoch for a cubic function.

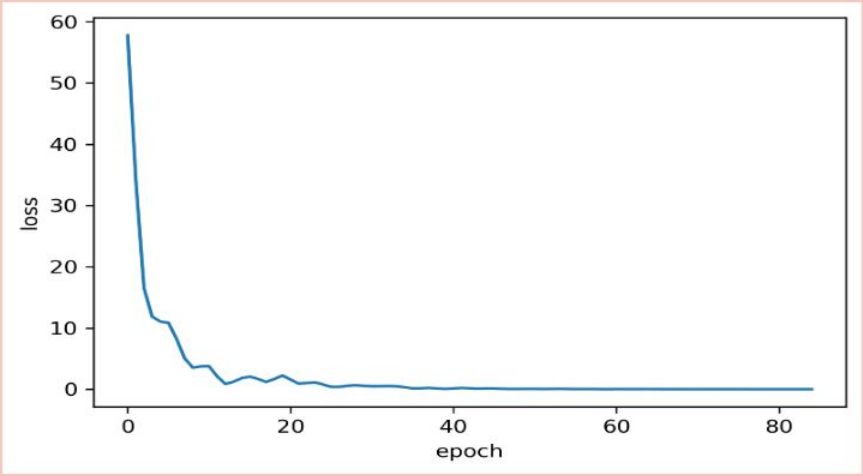


Figure 4. Loss at each epoch for a quadratic function.

In contrast, the loss for thequadratic function approached the optimal value at a significantly earlier period such that it stopped somewhere along the 80th to 85th epoch. This would mean that our model reached its optimal performance much earlier which suggests, it believes that it has already maxed the accuracy (with respect to the threshold of 0.01) for replicating the quadratic function.

Neural Networks for Curve-Fitting

As to visualize the accuracy of our results, the following images are the overlap of the actual functions and the neural network's attempt at reconstructing the functions. Although fairly accurate, further optimization can be performed to increase it even further.

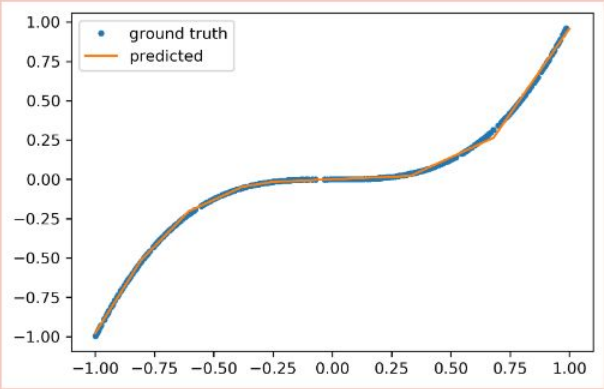


Figure 5. Curve fitting for a cubic function.

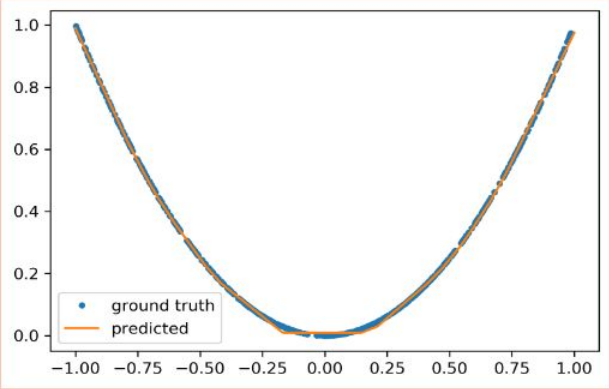


Figure 6. Curve fitting for a quadratic function.

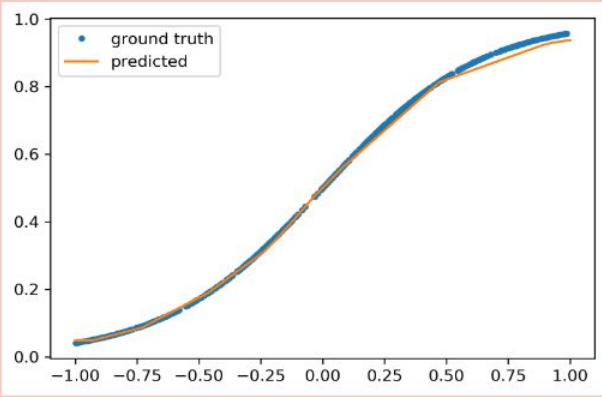


Figure 7. Curve fitting for a sigmoid function.

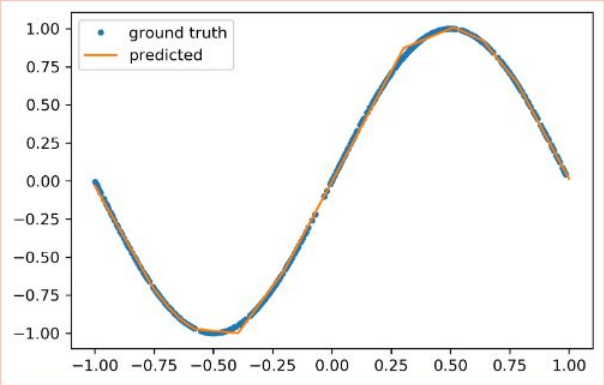


Figure 8. Curve fitting for a sin function.

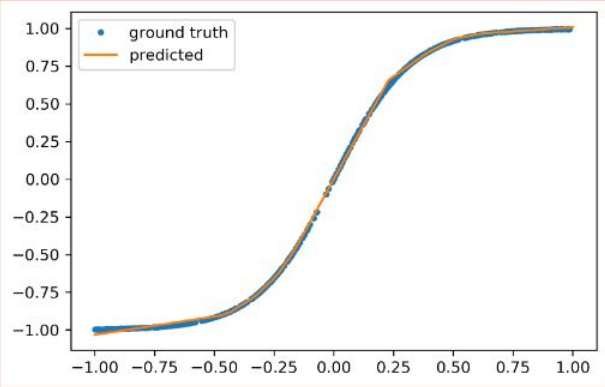


Figure 8. Curve fitting for a hyperbolic tangent function.

Neural Networks for Fruit-Classification

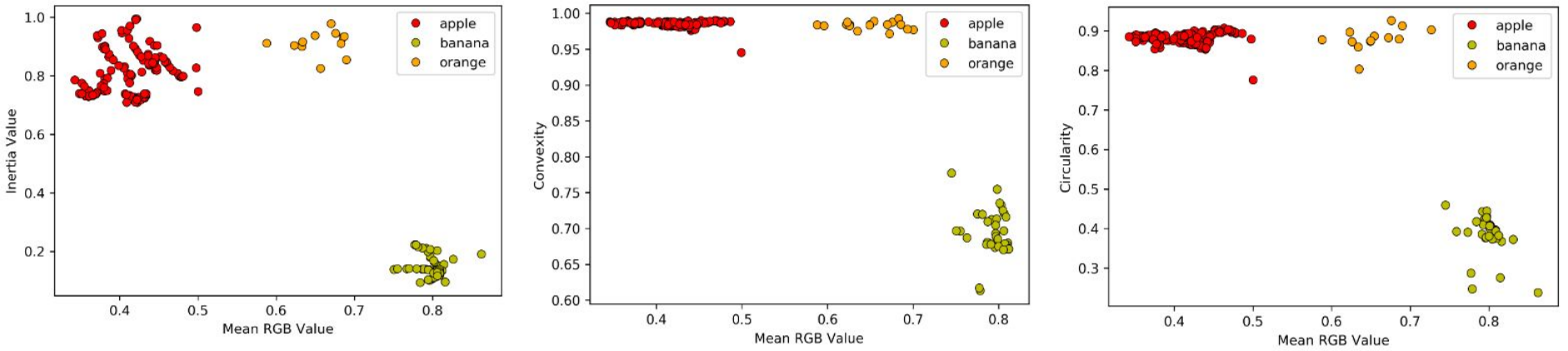


Figure 9. Feature space of our dataset combined with the Fruits360 dataset. The following pairs of features were observed for each fruit: Mean RGB pixel value and Inertia Ratio, Mean RGB pixel value and Convexity, and Mean RGB pixel value and Circularity. I ensured that the dataset had no overlapping features as this would results to inconsistencies within the network. Network can further be improved by accounting for such overlaps.

Another application of neural networks are for image classification. In this section , I present my results for my attempt at building a neural network for differentiating between the fruit dataset that I have been working on for the past few activities. Similar to the perceptron activity, I extracted various features which I will be using as an input to the network. I split my dataset into 50% for training and 50% for testing as to prevent any bias or mishandling for re-using training data for testing. I reflected Kenneth Domingo's method of using a factor of 1/10 for subsequent layers in the network such that the architecture of the network is constructed as: 193-19-1-3, where there are 3 output nodes to reflect the three different classes.

Neural Networks for Fruit-Classification

I designed my neural network such that each forward pass with respect to the hidden layers activates using the ReLU function, with the last layer as an exemption. The final hidden layer is activated using the sigmoid function,

```
def Sigmoid(Z):
    return 1/(1+np.exp(-Z))
```

The final layer was then activated using a softmax function which normalizes the value into a probability distribution consisting N probabilities that signify sums of the exponential of each input. With the aim of minimizing the mean-squared-error, our network's overall performance for each pair of features can be observed in the following slide,

Neural Networks for Fruit-Classification

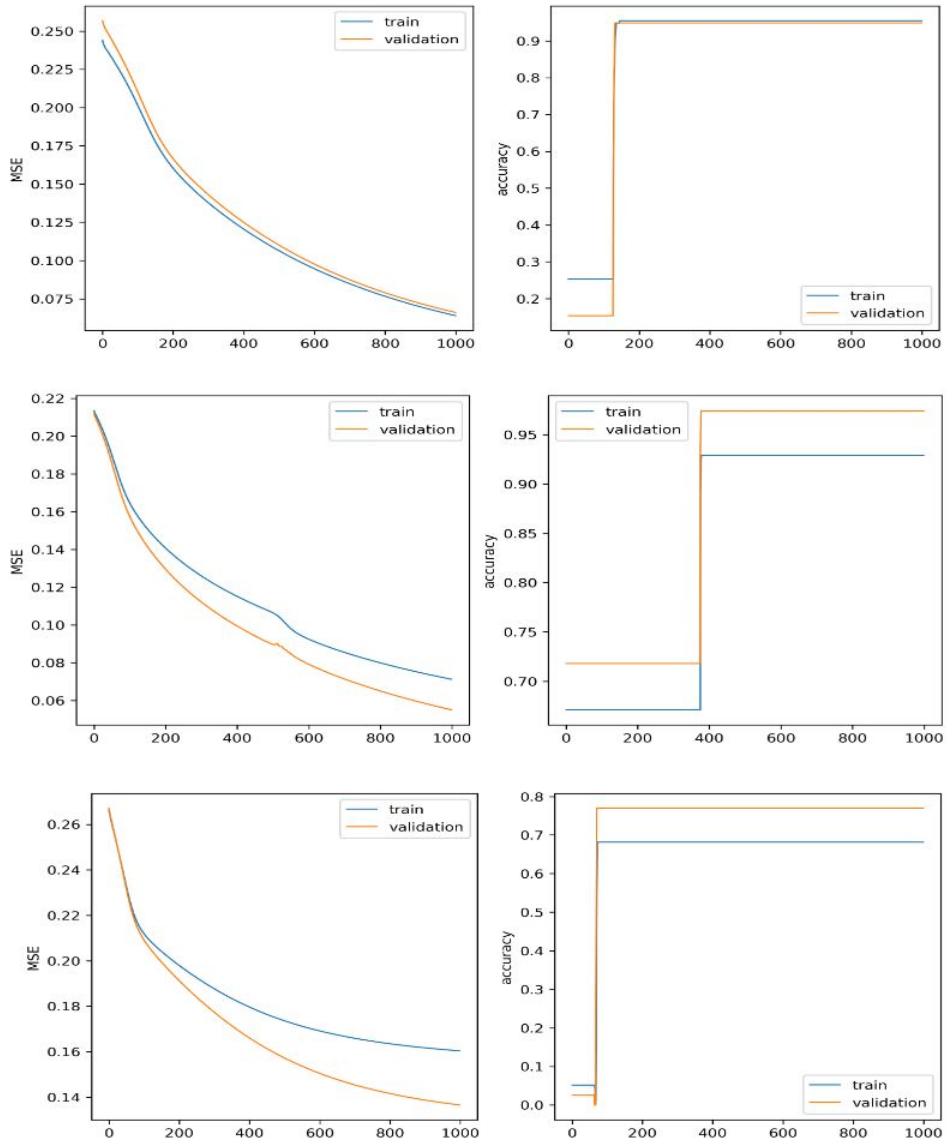


Figure 10. Mean-square error vs epoch and accuracy of results vs epoch of each feature combination. The generated results of our neural network was fairly accurate as it had followed a distinct trend for decreasing MSE through time. Although fair performance was observed for the training of mean RGB pixel vs circularity as it had a more relaxed slope for decreasing MSE and comparatively lower accuracy (70%) for the training. In contrast, the two other feature combination fared fairly well, scoring an average of 92% for mean RGB pixel vs convexity and an average of 91% for mean RGB pixel vs inertia ratio. Such deviations can further be optimized by increasing the number of epochs as well as balancing the number of datapoints for each class.

Applied Physics 186
Activity 17

QUALITY OF PRESENTATION: 5/5
TECHNICAL CORRECTNESS: 5/5
INITIATIVE:2

Reference for Fruit360 Dataset:

H. Muresan and M. Oltean, Fruit recognition from images using deep learning, *Acta Univ. Sapientiae, Informatica* **10**, 26 (2018).