Question 1

Description: The neural network or the artificial neural network is an algorithm to emulate the biological neuron brain cells. Similar to the brai cells, the neural networks consists of nodes in the hidden layer which individually contribute to the desired output. It can be used for complex analysis such as image recognition such as the following case.[1]
Reference: Wikipedia. (2020). Artificial neural network. [online] Available at: https://simple.wikipedia.org/wiki/Artificial_neural_network (https://simple.wikipedia.org/wiki/Artificial_neural_network) [Accessed 4 Mar. 2020].

Anitha Govindaraju(19230254) Jeyavignesh(19231993)

In [1]:

```
# Import statements
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [2]:

```
#Activation function
def sigmoid (t):
    return 1/(1 + np.exp(-t))

# Derivative of Sigmoid Function
def derivatives_sigmoid(t):
    return t * (1 - t)
```

In [0]:

```
#Anitha Govindaraju
# Method to initialize the values
def initvalues(X, hiddenlayer_neurons):
  """
  This method is used to initialize the initial weights and parameters for the hidden l
ayers, output and the input layer
  Parameters:
  input_neurons = input layer
  op_neurons = output layer
  weights_hidden = weights for the hidden layer
  bias_hidden = bias for the hidden layer
  weights_out = weights for the op layer
  bias_out = bias for the output layer
  """
  input_neurons = X.shape[1]
  op_neurons = 1
  weights_hidden = np.random.uniform(size=(input_neurons,hiddenlayer_neurons)) - 0.5
  bias_hidden = np.random.uniform(size=(1,hiddenlayer_neurons)) - 0.5
  weights_out = np.random.uniform(size=(hiddenlayer_neurons,op_neurons)) - 0.5
  bias_out = np.random.uniform(size=(1,op_neurons)) - 0.5
  print(weights_out.shape)
  return weights_hidden, bias_hidden, weights_out, bias_out
```

In [0]:

```
#Jeyavignesh
# Method to implement feed-forward neural nets
def feed_forward_neuralnet(X, weights_hidden, bias_hidden, weights_out, bias_out):
  """
  This method is to implement the feed forward neural net propagation.
  Parameters:
  hidden_input_layer = Hidden layer of input
  output_layer = Calculations for the output layer which includes the bias
  output = final weights
  """
  hidden_input_layer1 = np.dot(X,weights_hidden)
  hidden_input_layer = hidden_input_layer1 + bias_hidden
  hiddenlayer_activation = sigmoid(hidden_input_layer)
  output_1 = np.dot(hiddenlayer_activation,weights_out)
  output_layer = output_1 + bias_out
  output = sigmoid(output_layer)
  return output, hiddenlayer_activation
```

In [0]:

```
#Anitha Govindaraju
# Method to implement back propagation
def back_propagation(X, output, hiddenlayer_activations, weights_out, lr, bias_out, wei
ghts_hidden, bias_hidden, y):
  """
  This method is used to calculate the new updated weights by calculating the error
  Parameter:
  error = Gives the error between the actual output and the calculated output
  slope_hidden = Uses the derivative of sigmoid function to calculate the error rate an
d update the weights
  """
  error = y - output
  #print(error)
  slope_op = derivatives_sigmoid(output)
  slope_hidden = derivatives_sigmoid(hiddenlayer_activations)
  d_output = error * slope_op
  error_hidden_layer = d_output.dot(weights_out.T)
  d_hidden = error_hidden_layer * slope_hidden
  weights_out += hiddenlayer_activations.T.dot(d_output) * lr
  bias_out += np.sum(d_output, axis=0, keepdims=True) * lr
  weights_hidden += X.T.dot(d_hidden) * lr
  bias_hidden += np.sum(d_hidden, axis=0,keepdims=True) * lr
    # print("Weight output:",weights_out)
    # print("Bias output:",bias_out)
    # print("weights hidden:",weights_hidden)
    # print("Bias hidden",bias_hidden)
  return weights_out, bias_out, weights_hidden, bias_hidden
```

In [0]:

```
#Jeyavignesh
# Method which calculates the updated weights for given epochs
def neural_nets(X, y, hiddenlayer_neurons, epoch= 10000, lr = 0.01):
  """
  This method is used to train the neural nets and update the weights using the back pr
opagation.
  Parameters:
  weights_hidden = weights for the hidden layer
  bias_hidden = bias for the hidden layer
  weights_out = weights for the op layer
  bias_out = bias for the output layer
  """
  weights_hidden, bias_hidden, weights_out, bias_out = initvalues(X, hiddenlayer_neuron
s)
  for i in range(epoch):
      output, hiddenlayer_activations = feed_forward_neuralnet(X, weights_hidden, bias_
hidden, weights_out, bias_out)
      weights_out, bias_out, weights_hidden, bias_hidden = back_propagation(X, output,
hiddenlayer_activations, weights_out, lr, bias_out, weights_hidden, bias_hidden, y)
  #print("Output:", output)
  return weights_out, bias_out, weights_hidden, bias_hidden
```

In [0]:

```
#Anitha Govindaraju
# Method to calculate the output values by setting a threshold limit
def threshold(y):
  """
  This method is used to calculate the output y as 0 or 1 by setting a threshold limit
  Parameters:
  y_thershold = List containing the ouputs as either 0s or 1s
  """
  y_threshold = []
  for i in range(len(y)):
      if y[i] < 0.5:
          y_threshold.append(0)
      else:
          y_threshold.append(1)
  return y_threshold
```

Question 2:

In [26]:

```
#Reading the sample CSV file to plot the data
data = pd.read_csv(r"C:\Users\anith\OneDrive\Desktop\Semester_2\Deep_learning\Assignmen
t1\circles500.csv")
X = data.drop(['Class'], axis=1).values
y = np.array(data['Class'].values)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.10, random_stat
e = 42)
y_train = np.reshape(y_train, (len(y_train), 1))
y_test = np.reshape(y_test, (len(y_test), 1))
```

In [386]:

```
# Training the neural net for the initial model
update_wout, updated_bout, updated_wh, updated_bh = neural_nets(X_train, y_train, 3)
```

```
(3, 1)
```

In [0]:

```
#Jeyavignesh
# Method to predict the output using updated weights from the trained model
def prediction(X, weights_out, bias_out, weights_hidden, bias_hidden):
  """
  Parameters:
  output = gives the output weights between the range of 0 to 1 in decimals
  y_pred = uses the ouput to calculate the threshold and update the ouput either as 0 o
r 1
  """
  output, hiddenlayer_activations = feed_forward_neuralnet(X, weights_hidden, bias_hidd
en, weights_out, bias_out)
  y_pred = threshold(output)
  return y_pred
```

In [388]:

```
# Predicting the ouput for Circles data using the hold out set
y_pred = prediction(X_test, update_wout, updated_bout, updated_wh, updated_bh)
#print(y_pred)
Accuracy = accuracy_score(y_test, y_pred)
print(Accuracy)
```

```
0.98
```

Observations

Test Run 1: Learning rate: 0.01, Epoch: 100 Accuracy: 44

Test Run 2: Learning rate: 0.01, Epoch: 1000 Accuracy: 44

Test Run 3: Learning rate: 0.01, Epoch: 10000 Accuracy: 98

Question 3:

Training the big data set

In [0]:

```
# Referece: Dr. Michael Madden
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

In [0]:

```
#Anitha Govindaraju
import matplotlib.pyplot as plt
# Reference: Dr. Michael Madden
def visualise(data, index):
    # MM Jan 2019: Given a CIFAR data nparray and the index of an image, display the im
age.
    # Note that the images will be quite fuzzy looking, because they are low res (32x3
2).
    picture = data[index]
    # Initially, the data is a 1D arwray of 3072 pixels; reshape it to a 3D array of 3x
32x32 pixels
    # Note: after reshaping like this, you could select one colour channel or average t
hem.
    picture.shape = (3,32,32)

    picture = picture.transpose([1, 2, 0])

    #https://www.codementor.io/@innat_2k14/image-data-analysis-using-numpy-opencv-part-
1-kfadbafx6
    gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])
    gray = gray(picture)

    #plt.imshow(gray, cmap = plt.get_cmap(name = 'gray'))
#     plt.imshow(gray)
#     plt.show()
    return gray
```

In [391]:

```
#Jeyavignesh
batch1 = unpickle(r'data_batch_1')
print("Number of items in the batch is", len(batch1))

# Display all keys, so we can see the ones we want
print('All keys in the batch:', batch1.keys())
# Extracting data with Deer and dog alone for X and y datasets
data = batch1[b'data']
labels = batch1[b'labels']

meta = unpickle(r'batches.meta')
names = meta[b'label_names']
print(names)
deer_dog_list = []

for i in range(len(data)):
    if labels[i] in [4,5]:
        deer_dog_list.append(i)

DD_data = data[deer_dog_list, :]
label_array = np.array(labels)
DD_labels = label_array[deer_dog_list]

print(len(DD_data))
print(len(DD_labels))
```

```
Number of items in the batch is 4
All keys in the batch: dict_keys([b'batch_label', b'labels', b'data', b'fi
lenames'])
[b'airplane', b'automobile', b'bird', b'cat', b'deer', b'dog', b'frog',
b'horse', b'ship', b'truck']
1936
1936
```

In [0]:

```
#Anitha Govindaraju
# Converting the image to greyscale pattern
input_img = []
output_label = []
for i in range (len(DD_data)):
#for i in range (1, 20):
    grey = visualise(DD_data, i)
    input_img.append(grey.flatten())
    label = DD_labels[i]
    output_label.append(label)
```

In [393]:

```
input_img = np.array(input_img)
print(input_img.shape)
output_label = np.array(output_label)
```

```
(1936, 1024)
```

In [0]:

```
# Generating input and output data
X, y = input_img, output_label
# Normalizing the data for X
X = X/255
# Replacing labels for Dog and deer with 0 and 1
y[y==4] = 0
y[y==5] = 1
```

Train Test split

In [0]:

```
# Train, test split for validation
X_train_CIF, X_test_CIF, y_train_CIF, y_test_CIF = train_test_split(X, y, test_size =
0.10, random_state = 42)
# y_train_CIF = np.reshape(y_train_CIF, (len(y_train_CIF), 1))
y_test_CIF = np.reshape(y_test_CIF, (len(y_test_CIF), 1))
y_train_CIF = y_train_CIF.reshape(-1,1)
```

In [396]:

```
# Training the neural network with X_train- training dataset
img_update_wout, img_updated_bout, img_updated_wh, img_updated_bh = neural_nets(X_train
_CIF,y_train_CIF,1024, epoch= 1000, lr = 1e-08)
```

```
(1024, 1)
```

In [397]:

```
# Using test set to predict the output of the model
y_pred_img = prediction(X_test_CIF,img_update_wout, img_updated_bout, img_updated_wh, i
mg_updated_bh)
# Calculating accuracy using scikit accuracy metrics comparing predicted values and the
actual prediction
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img)
print(Accuracy_img)
```

```
0.5154639175257731
```

In [398]:

```
img_update_wout, img_updated_bout, img_updated_wh, img_updated_bh = neural_nets(X_train
_CIF,y_train_CIF,1024, epoch= 100, lr = 1e-08)
```

```
(1024, 1)
```

In [399]:

```
y_pred_img = prediction(X_test_CIF,img_update_wout, img_updated_bout, img_updated_wh, i
mg_updated_bh)
# Calculating accuracy using scikit accuracy metrics comparing predicted values and the
actual prediction
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img)
print(Accuracy_img)
```

```
0.5721649484536082
```

Observation

The model performed poorly when the learning rate was set to 0.01. Over 20 test runs were done adjusting the learning rate and the epoch. The first real change in weights occured when the learning rate was set to 1e-08. The previous runs yeilded a prediction which contained same label. Sharing details of a few test runs.

Test Run 1: Learning rate: 0.01, Epoch: 100 Accuracy: 49

Test Run 2: Learning rate: 1e-07, Epoch: 1200 Accuracy: 40

Test Run 3: Learning rate: 1e-07, Epoch: 1100 Accuracy: 53

Test Run 4: Learning rate: 0.01, Epoch: 1000 Accuracy: 46

Part 4a: Jeyavignesh Rajendran

Explanation:

I have added an additional hidden layer and calculated the neurons just to prevent overfitting the data.

$Nh=Ns(\alpha*(Ni+No))$

Ni = number of input neurons.

No = number of output neurons.

Ns = number of samples in training data set.

α = an arbitrary scaling factor usually 2-10.[2]

The optimised model has all the methods re-written to fit in the extra hidden layer.

Code runs were conducted for all the batch files and 1 batch file for the big dataset. The same code was run for the small dataset which yeilded 100 percent accuracy.

For the big dataset, like the previous model, first change was detected when the learning rate was set to 1e-08. The model performed well when the epoch was set to 10000. A maximum accuracy of 61 percent was achieved for epoch 10000. For smaller runs, the model yeilded an accuracy around 54.

Additionally, I tried changing the activation function in hidden layers and the output layer to determine the determine the best model, but the results were significantly low.

Reference:

H. and Hyndman, R. (2020). How to choose the number of hidden layers and nodes in a feedforward neural network?. [online] Cross Validated. Available at: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw (https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw) [Accessed 4 Mar. 2020].

In [0]:

```
def initvalues_optimised(X, hiddenlayer_neuron1, hiddenlayer_neuron2):
  """
  This method is used to initialize the initial weights and parameters for the hidden l
ayers, output and the input layer
  Parameters:
  input_neurons = input layer
  op_neurons = output layer
  weights_hidden = weights for the hidden layer
  bias_hidden = bias for the hidden layer
  weights_out = weights for the op layer
  bias_out = bias for the output layer
  """
  input_neurons = X.shape[1]
  op_neurons = 1
  weights_hidden = np.random.uniform(size=(input_neurons,hiddenlayer_neuron1)) - 0.5
  bias_hidden = np.random.uniform(size=(1,hiddenlayer_neuron1)) - 0.5
  weights_hidden2 = np.random.uniform(size=(hiddenlayer_neuron1,hiddenlayer_neuron2)) -
0.5
  bias_hidden2 = np.random.uniform(size=(1,hiddenlayer_neuron2)) - 0.5
  weights_out = np.random.uniform(size=(hiddenlayer_neuron2,op_neurons)) - 0.5
  bias_out = np.random.uniform(size=(1,op_neurons)) - 0.5
  print(weights_hidden.shape)
  print(weights_hidden2.shape)
  return weights_hidden, bias_hidden,weights_hidden2, bias_hidden2, weights_out, bias_o
ut
```

In [0]:

```
# Method to implement feed-forward neural nets
def feed_forward_neuralnet_op(X, weights_hidden, bias_hidden,weights_hidden2, bias_hidd
en2, weights_out, bias_out):
  """
  This method is to implement the feed forward neural net propagation.
  Parameters:
  hidden_input_layer = Hidden layer of input
  output_layer = Calculations for the output layer which includes the bias
  output = final weights
  """
  hidden_input_layer1 = np.dot(X,weights_hidden)
  hidden_input_layer = hidden_input_layer1 + bias_hidden
  hiddenlayer_activation = sigmoid(hidden_input_layer)
  hidden_input_layer2 = np.dot(hidden_input_layer1,weights_hidden2)
  hidden_input_layer3 = hidden_input_layer2 + bias_hidden2
  hiddenlayer_activation2 = sigmoid(hidden_input_layer3)
  output_1 = np.dot(hiddenlayer_activation2,weights_out)
  output_layer = output_1 + bias_out
  output = sigmoid(output_layer)
  return output, hiddenlayer_activation, hiddenlayer_activation2
```

In [0]:

```
# Method to implement back propagation
def back_propagation_op(X, output, hiddenlayer_activation1,hiddenlayer_activation2,
                        weights_out, lr, bias_out, weights_hidden1, bias_hidden1,weight
s_hidden2, bias_hidden2, y):
  """
  This method is used to calculate the new updated weights by calculating the error
  Parameter:
  error = Gives the error between the actual output and the calculated output
  slope_hidden = Uses the derivative of sigmoid function to calculate the error rate an
d update the weights
  """
  error = y - output
  #print(error)
  slope_op = derivatives_sigmoid(output)
  slope_hidden2 = derivatives_sigmoid(hiddenlayer_activation2)
  #print(hiddenlayer_activation1.shape)
  slope_hidden1 = derivatives_sigmoid(hiddenlayer_activation1)
  #print(slope_hidden1.shape)
  d_output = error * slope_op
  error_hidden_layer2 = d_output.dot(weights_out.T)
  d_hidden2 = error_hidden_layer2 * slope_hidden2
  error_hidden_layer1 = d_hidden2.dot(weights_hidden2.T)
  d_hidden1 = error_hidden_layer1 * slope_hidden1
  weights_out += hiddenlayer_activation2.T.dot(d_output) * lr
  bias_out += np.sum(d_output, axis=0, keepdims=True) * lr
  weights_hidden2 += hiddenlayer_activation2.T.dot(d_hidden2) * lr
  bias_hidden2 += np.sum(d_hidden2, axis=0,keepdims=True) * lr
  weights_hidden1 += X.T.dot(d_hidden1) * lr
  bias_hidden1 += np.sum(d_hidden1, axis=0,keepdims=True) * lr
  return weights_out, bias_out, weights_hidden1, bias_hidden1, weights_hidden2, bias_hi
dden2
```

In [0]:

```
# Method which calculates the updated weights for given epochs
def neural_nets_op(X, y, hiddenlayer_neurons, epoch= 10000, lr = 0.01):
  """
  This method is used to train the neural nets and update the weights using the back pr
opagation.
  Parameters:
  weights_hidden = weights for the hidden layer
  bias_hidden = bias for the hidden layer
  weights_out = weights for the op layer
  bias_out = bias for the output layer
  """
  weights_hidden, bias_hidden,weights_hidden2, bias_hidden2, weights_out, bias_out = in
itvalues_optimised(X, hiddenlayer_neurons, hiddenlayer_neurons)
  for i in range(epoch):
      #output, hiddenlayer_activations = feed_forward_neuralnet(X, weights_hidden, bias
_hidden, weights_out, bias_out)
      #weights_out, bias_out, weights_hidden, bias_hidden = back_propagation(X, output,
hiddenlayer_activations, weights_out, lr, bias_out, weights_hidden, bias_hidden, y)
  #print("Output:", output)
      output, hiddenlayer_activation1, hiddenlayer_activation2 = feed_forward_neuralne
t_op(X, weights_hidden, bias_hidden,weights_hidden2, bias_hidden2, weights_out, bias_ou
t)
      weights_out, bias_out, weights_hidden1, bias_hidden1, weights_hidden2, bias_hidd
en2 = back_propagation_op(X, output, hiddenlayer_activation1,hiddenlayer_activation2, w
eights_out, lr, bias_out, weights_hidden, bias_hidden,weights_hidden2, bias_hidden2, y)
  return weights_out, bias_out, weights_hidden1, bias_hidden1, weights_hidden2, bias_hi
dden2
```

In [421]:

```
update_wout, updated_bout, updated_wh1, updated_bh1, updated_wh2, updated_bh2  = neural
_nets_op(X_train, y_train, 4)
```

```
(2, 4)
(4, 4)
```

In [0]:

```
def prediction_op(X, weights_out, bias_out, weights_hidden1, bias_hidden1,weights_hidde
n2, bias_hidden2):
  """
  Parameters:
  output = gives the output weights between the range of 0 to 1 in decimals
  y_pred = uses the ouput to calculate the threshold and update the ouput either as 0 o
r 1
  """
  #output, hiddenlayer_activations = feed_forward_neuralnet(X, weights_hidden, bias_hid
den, weights_out, bias_out)
  output, hiddenlayer_activation, hiddenlayer_activation2 = feed_forward_neuralnet_op(X
, weights_hidden1, bias_hidden1,weights_hidden2, bias_hidden2, weights_out, bias_out)
  y_pred = threshold(output)
  return y_pred
```

In [423]:

```
# Predicting the ouput for Circles data using the hold out set
y_pred = prediction_op(X_test, update_wout, updated_bout, updated_wh1, updated_bh1,upda
ted_wh2, updated_bh2)
#print(y_pred)
Accuracy = accuracy_score(y_test, y_pred)
print(Accuracy)
```

```
1.0
```

In [424]:

```
hidden_neurons_size = len(X_train_CIF) / (2 * (32 + 2 ))
print(hidden_neurons_size)
print(len(X_train_CIF))
```

```
25.61764705882353
1742
```

In [428]:

```
img_weights_out, img_bias_out, img_weights_hidden1, img_bias_hidden1, img_weights_hidde
n2, img_bias_hidden2 = neural_nets_op(X_train_CIF,y_train_CIF,1024, epoch= 100, lr = 1e
-08)
```

```
(1024, 1024)
(1024, 1024)
```

In [429]:

```
# Using test set to predict the output of the model
y_pred_img_op = prediction_op(X_test_CIF, img_weights_out, img_bias_out, img_weights_hi
dden1, img_bias_hidden1,img_weights_hidden2, img_bias_hidden2)
# Calculating accuracy using scikit accuracy metrics comparing predicted values and the
actual prediction
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img_op)
print(Accuracy_img)
```

```
0.5412371134020618
```

In [430]:

```
np.unique(y_pred_img_op)
```

Out[430]:

```
array([0, 1])
```

In [0]:

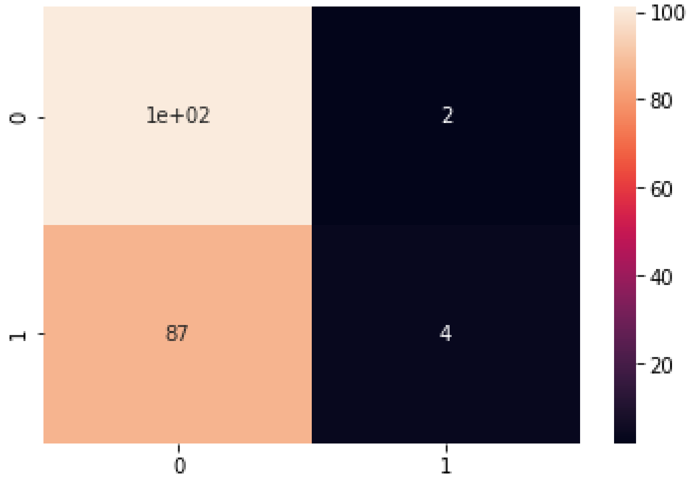
```
img_weights_out, img_bias_out, img_weights_hidden1, img_bias_hidden1, img_weights_hidde
n2, img_bias_hidden2 = neural_nets_op(X_train_CIF,y_train_CIF,1024, epoch= 10000, lr =
1e-08)
```

In [0]:

```
# Using test set to predict the output of the model
y_pred_img_op = prediction_op(X_test_CIF, img_weights_out, img_bias_out, img_weights_hi
dden1, img_bias_hidden1,img_weights_hidden2, img_bias_hidden2)
# Calculating accuracy using scikit accuracy metrics comparing predicted values and the
actual prediction
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img_op)
print(Accuracy_img)
```

In [433]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cfm = confusion_matrix(y_test_CIF, y_pred_img_op)
sns.heatmap(cfm, annot = True);
```



In [0]:

```
X  = list()
Y = list()
y_labs = {'deer':0 , 'dog':1}
for _i in range(1,6):
        batch1 = unpickle('data_batch_{0}'.format(_i))
        data = batch1[b'data']
        labels = batch1[b'labels']
        for i , image in enumerate(data):
                _label = names[labels[i]].decode("utf-8")
                if  _label in ['deer' ,'dog']:
                        image.shape = (3,32,32)
                        image = image.transpose([1, 2, 0])
                        B, G, R     = image[:, :, 0], image[:, :, 1], image[:, :, 2] # F
or RGB image
                        grey_scaled = 0.2125 * R + 0.7154 * G + 0.0721 * B
                        grey_scaled = grey_scaled /255
                        grey_scaled = list(grey_scaled.flatten())
                        X.append(grey_scaled)
                        Y.append(y_labs[_label])

X = np.array(X)
Y = np.array(Y)
```

Sample run for full dataset with 10 epochs yeilded different results for the prediction class. With 10000 model performed significantly well.

In [0]:

```
X_train_CIF, X_test_CIF, y_train_CIF, y_test_CIF = train_test_split(X, Y, test_size =
0.10, random_state = 42)
# y_train_CIF = np.reshape(y_train_CIF, (len(y_train_CIF), 1))
y_test_CIF = np.reshape(y_test_CIF, (len(y_test_CIF), 1))
y_train_CIF = y_train_CIF.reshape(-1,1)
```

In [448]:

```
img_weights_out, img_bias_out, img_weights_hidden1, img_bias_hidden1, img_weights_hidde
n2, img_bias_hidden2 = neural_nets_op(X_train_CIF,y_train_CIF,1024, epoch= 10, lr = 1e-
08)
```

```
(1024, 1024)
(1024, 1024)
```

In [449]:

```
# Using test set to predict the output of the model
y_pred_img_op = prediction_op(X_test_CIF, img_weights_out, img_bias_out, img_weights_hi
dden1, img_bias_hidden1,img_weights_hidden2, img_bias_hidden2)
# Calculating accuracy using scikit accuracy metrics comparing predicted values and the
actual prediction
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img_op)
print(Accuracy_img)
```

```
0.434
```

In [450]:

```
np.unique(y_pred_img_op)
```

Out[450]:

```
array([0, 1])
```

In [0]:

```
img_weights_out, img_bias_out, img_weights_hidden1, img_bias_hidden1, img_weights_hidde
n2, img_bias_hidden2 = neural_nets_op(X_train_CIF,y_train_CIF,1024, epoch= 10000, lr =
1e-08)
```

In [ ]:

```
# Using test set to predict the output of the model
y_pred_img_op = prediction_op(X_test_CIF, img_weights_out, img_bias_out, img_weights_hi
dden1, img_bias_hidden1,img_weights_hidden2, img_bias_hidden2)
# Calculating accuracy using scikit accuracy metrics comparing predicted values and the
actual prediction
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img_op)
print(Accuracy_img)
```

Part 4B: Anitha Govindaraju(19230254)

Optimization:

1. I have implemented a new activation function tanh and its derivative which is a non linear function. The intervals range from -1 to 1. The tanh activation function has a gradient stronger than sigmoid.
2. I have implemented logic for Adaptive learning rate to extract the optimum learning rate based on the number of epochs provided. This rate automatically adapts are generates higher accuray naturally.

Observations:

After implementing the above mentioned enhancements, the accuracy rate for a simple dataset provided in task2 has increased to 98%, the accuracy for image classification task dataset has increased to 62% from 51%.

In [114]:

```
# Import statements
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#Reading the sample CSV file to plot the data
data = pd.read_csv(r"C:\Users\anith\OneDrive\Desktop\Semester_2\Deep_learning\Assignmen
t1\circles500.csv")
X = data.drop(['Class'], axis=1).values
y = np.array(data['Class'].values)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.10, random_stat
e = 42)
y_train = np.reshape(y_train, (len(y_train), 1))
y_test = np.reshape(y_test, (len(y_test), 1))
```

In [115]:

```
#Activation function
def tanh(X):
    return (np.exp(X)-np.exp(-X))/(np.exp(X)+np.exp(-X))

# Derivative of tanh Function
def derivative_of_tanh(X):
    return 1-(tanh(X)**2)
```

In [116]:

```
# Method to initialize the values
def initvalues_optimization(X, hiddenlayer_neurons):
    """
    This method is used to initialize the initial weights and parameters for the hidden l
ayers, output and the input layer
    Parameters:
    input_neurons = input layer
    op_neurons = output layer
    weights_hidden = weights for the hidden layer
    bias_hidden = bias for the hidden layer
    weights_output = weights for the op layer
    bias_output = bias for the output layer
    """
    input_neurons = X.shape[1]
    op_neurons = 1
    weights_hidden = np.random.uniform(size=(input_neurons,hiddenlayer_neurons)) - 0.5
    bias_hidden = np.random.uniform(size=(1,hiddenlayer_neurons)) - 0.5
    weights_output = np.random.uniform(size=(hiddenlayer_neurons,op_neurons)) - 0.5
    bias_output = np.random.uniform(size=(1,op_neurons)) - 0.5
    return weights_hidden, bias_hidden, weights_output, bias_output
```

In [117]:

```
# Method to implement feed-forward neural nets
def feed_forward_neuralnet_optimization(X, weights_hidden, bias_hidden, weights_output,
bias_output):
    """
    This method is to implement the feed forward neural net propagation.
    Parameters:
    input_hidden_layer_P = Hidden layer of input
    output_layer = Calculations for the output layer which includes the bias
    output = final weights
    """
    input_hidden_layer_P = np.dot(X,weights_hidden)
    input_hidden_layer = input_hidden_layer_P + bias_hidden
    input_hidden_activation = tanh(input_hidden_layer)
    output_P = np.dot(input_hidden_activation,weights_output)
    output_layer = output_P + bias_output
    output = tanh(output_layer)
    return output, input_hidden_activation
```

In [118]:

```
# Method to implement back propagation
def back_propagation_optimization(X, output, input_hidden_activation, weights_output, l
r, bias_output, weights_hidden, bias_hidden, y):
  """
  This method is used to calculate the new updated weights by calculating the error
  Parameter:
  error = Gives the error between the actual output and the calculated output
  slope_hidden = Uses the derivative of sigmoid function to calculate the error rate an
d update the weights
  """
  error = y - output

  der_output = derivative_of_tanh(output)
  der__hidden = derivative_of_tanh(input_hidden_activation)
  der_output = error * der_output
  error_hidden_layer = der_output.dot(weights_output.T)
  der_hidden = error_hidden_layer * der__hidden
  weights_output += input_hidden_activation.T.dot(der_output) * lr
  bias_output += np.sum(der_output, axis=0, keepdims=True) * lr
  weights_hidden += X.T.dot(der_hidden) * lr
  bias_hidden += np.sum(der_hidden, axis=0,keepdims=True) * lr

  return weights_output, bias_output, weights_hidden, bias_hidden
```

In [119]:

```
# Method to predict the output using updated weights from the trained model
def prediction_optimization(X, weights_output, bias_output, weights_hidden, bias_hidden
):
  """
  Parameters:
  output = gives the output weights between the range of 0 to 1 in decimals
  y_pred = uses the ouput to calculate the threshold and update the ouput either as 0 o
r 1
  """
  output, input_hidden_activation = feed_forward_neuralnet_optimization(X, weights_hidd
en, bias_hidden, weights_output, bias_output)
  y_pred = threshold_optimization(output)
  return y_pred
```

In [120]:

```
# Method to calculate the output values by setting a threshold limit
def threshold_optimization(y):
  """
  This method is used to calculate the output y as 0 or 1 by setting a threshold limit
  Parameters:
  y_thershold = List containing the ouputs as either 0s or 1s
  """
  y_threshold = []
  for i in range(len(y)):
      if y[i] < 0.5:
          y_threshold.append(0)
      else:
          y_threshold.append(1)
  return y_threshold
```

In [121]:

```
#[Reference: https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-ne
ural-networks-3fe03dad8594]
# Method which calculates the updated weights for given epochs
def neural_nets_optimization(X, y,X_test, y_test, hiddenlayer_neurons, epoch, lr):
    """
    This method is used to
    1. Update the learning rate using Adaptive learning rate method which will ultimately
increase the accuracy and
       return the optimum weights. This optimum weights will be used for predicting the a
ccuracy of test data.
    2. Train the neural nets and update the weights using the back propagation.
    Parameters:
    weights_hidden = weights for the hidden layer
    bias_hidden = bias for the hidden layer
    weights_output = weights for the op layer
    bias_output = bias for the output layer
    """
    accuracy_1 = 0
    weights_hidden, bias_hidden, weights_output, bias_output = initvalues_optimization(X,
hiddenlayer_neurons)
    for num in range(epoch):
          if num % 100 == 0:
              y_pred = prediction_optimization(X_test, weights_output, bias_output, weigh
ts_hidden, bias_hidden)
              Accuracy = accuracy_score(y_test, y_pred)
              decay = lr / epoch
              if accuracy_1 > Accuracy:
                  lr = lr * (1 / (1 + decay * epoch))
              else:
                  lr = lr / (1 / (1 + decay * epoch))
              accuracy_1 = Accuracy
          output, input_hidden_activation = feed_forward_neuralnet_optimization(X, weight
s_hidden, bias_hidden, weights_output, bias_output)
          weights_output, bias_output, weights_hidden, bias_hidden = back_propagation_opt
imization(X, output, input_hidden_activation, weights_output, lr, bias_output, weights_
hidden, bias_hidden, y)
    print("Optimum Learning rate is:", lr)
    print("Optimized Accuracy for trained data is:", Accuracy)
    return weights_output, bias_output, weights_hidden, bias_hidden
```

In [113]:

```
# Training the neural net for the initial model
update_wout, updated_bout, updated_wh, updated_bh = neural_nets_optimization(X_train, y
_train,X_test,y_test, 3, 100000, lr = 0.0001)
```

```
Optimum Learning rate is: 0.00011078977382657771
Optimized Accuracy for trained data is: 0.98
```

In [72]:

```
# Prediction step to calculate the optimum accuracy score using X_test
y_pred = prediction_optimization(X_test,update_wout, updated_bout, updated_wh, updated_
bh)
Accuracy = accuracy_score(y_test, y_pred)
print("Accuracy for Task2 dataset after optimisation:", Accuracy)
```

```
Accuracy for Task2 dataset after optimisation: 0.98
```

In [49]:

```
# Reference: Dr. Michael Madden
def unpickle_optimization(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

In [50]:

```
# [Reference: Dr. Michael Madden]
# [Refrence: #https://www.codementor.io/@innat_2k14/image-data-analysis-using-numpy-ope
ncv-part-1-kfadbafx6]

import matplotlib.pyplot as plt
def visualise_optimization(data, index):
    # MM Jan 2019: Given a CIFAR data nparray and the index of an image, display the im
age.
    # Note that the images will be quite fuzzy looking, because they are low res (32x3
2).

    picture = data[index]
    # Initially, the data is a 1D array of 3072 pixels; reshape it to a 3D array of 3x3
2x32 pixels
    # Note: after reshaping like this, you could select one colour channel or average t
hem.

    picture.shape = (3,32,32)
    picture = picture.transpose([1, 2, 0])

    #Code to convert the rgb images into blue scale.
    gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])
    gray = gray(picture)

    return gray
```

In [51]:

```
# [Reference: Dr. Michael Madden]
batch1 = unpickle_optimization(r'C:\Users\anith\OneDrive\Desktop\Semester_2\Deep_learni
ng\Assignment1\cifar-10-batches-py\data_batch_1')
print("Number of items in the batch is", len(batch1))

# Display all keys, so we can see the ones we want
print('All keys in the batch:', batch1.keys())
```

```
Number of items in the batch is 4
All keys in the batch: dict_keys([b'batch_label', b'labels', b'data', b'fi
lenames'])
```

In [52]:

```
# [Reference: Dr. Michael Madden]
# Extracting data with Deer and dog alone for X and y datasets
data = batch1[b'data']
labels = batch1[b'labels']

meta = unpickle_optimization(r'C:\Users\anith\OneDrive\Desktop\Semester_2\Deep_learning
\Assignment1\cifar-10-batches-py\batches.meta')
names = meta[b'label_names']

#code extract only deer and dog information
deer_dog_list = []
for i in range(len(data)):
    if labels[i] in [4,5]:
        deer_dog_list.append(i)

DD_data = data[deer_dog_list, :]
label_array = np.array(labels)
DD_labels = label_array[deer_dog_list]
```

In [53]:

```
# Converting the image to greyscale pattern
input_img = []
output_label = []
for i in range (len(DD_data)):
    grey = visualise_optimization(DD_data, i)
    input_img.append(grey.flatten())
    label = DD_labels[i]
    output_label.append(label)
```

In [54]:

```
#Converting list to nd numpy array
input_img = np.array(input_img)
output_label = np.array(output_label)
```

In [55]:

```
# Generating input and output data
X, y = input_img, output_label
```

In [56]:

```
# Normalizing the data for X
X = X/255
# Replacing labels for Dog and deer with 0 and 1
y[y==4] = 0
y[y==5] = 1
```

In [57]:

```
# Train, test split for validation
X_train_CIF, X_test_CIF, y_train_CIF, y_test_CIF = train_test_split(X, y, test_size =
0.10, random_state = 42)
y_test_CIF = np.reshape(y_test_CIF, (len(y_test_CIF), 1))
y_train_CIF = y_train_CIF.reshape(-1,1)
```

In [58]:

```
# Training the neural net for the initial model
img_update_wout, img_updated_bout, img_updated_wh, img_updated_bh = neural_nets_optimiz
ation(X_train_CIF,y_train_CIF,X_test_CIF, y_test_CIF,1024, epoch= 1000, lr = 0.0001)
```

```
Optimum Learning rate is: 0.00010006002801110384
Optimized Accuracy for trained data is: 0.6288659793814433
```

In [59]:

```
# Prediction step to calculate the optimum accuracy score using X_test
y_pred_img = prediction_optimization(X_test_CIF,img_update_wout, img_updated_bout, img_
updated_wh, img_updated_bh)
Accuracy_img = accuracy_score(y_test_CIF, y_pred_img)
print("Accuracy for Task3 image dataset after optimisation:",Accuracy_img)
```

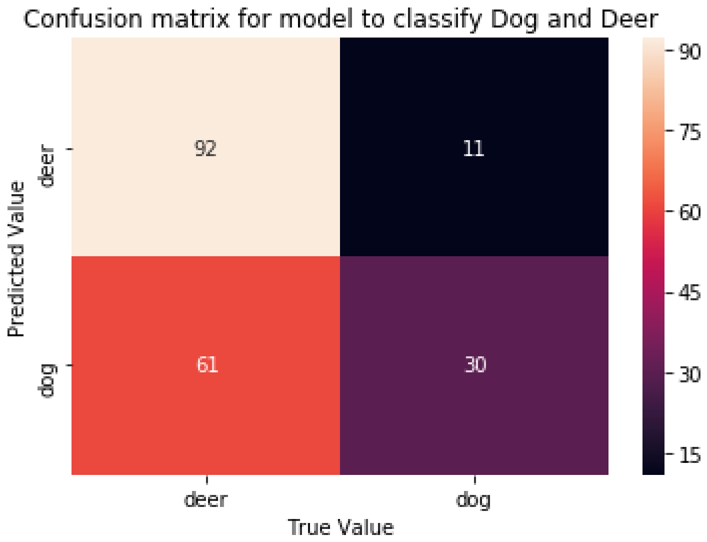
```
Accuracy for Task3 image dataset after optimisation: 0.6288659793814433
```

In [91]:

```
''' Confusion matrix for y_test and y_predict '''
from sklearn.metrics import confusion_matrix
import seaborn as sb
label = ['deer','dog']
con_matrix = confusion_matrix(y_test_CIF, y_pred_img)
Hm = sb.heatmap(con_matrix, annot = True, xticklabels = label, yticklabels = label);
Hm.set(xlabel='True Value', ylabel='Predicted Value')
Hm.set_title('Confusion matrix for model to classify Dog and Deer')
```

Out[91]:

```
Text(0.5, 1, 'Confusion matrix for model to classify Dog and Deer')
```



In [ ]: