CS3642-02

Jack Burns

001013319

Assignment #3

10/17/24



# Perceptron

## Problems:

* 4-Pixel Image: Decides if an image is bright or dark based on 4 pixel input. If it contains 2, 3 or 4 white pixels, it’s bright, otherwise it’s dark
* CMYK value: Decides if a color value is too oversaturated for printing. If the color value is over 240% the color is oversaturated.

## Statistics:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy (Training) | Accuracy (Testing) | Time Complexity |
| 4-pixel images | 100% | 50% |  |
| CMYK |  |  |  |

## Code:

Perceptron Class:

|  |
| --- |
| import random  class Perceptron:      def \_\_init\_\_(self, x, y, threshold):          self.x=x          #insert bias neuron at beginning of input array          self.x.insert(0,1)          self.y=y          self.threshold = threshold          self.w = []          self.epochs = 0          #initialize weights as random numbers -0.5 - .0.5          for i in range(len(x)):              self.w.append(random.randrange(-5,5)/10 )          self.alpha = 0.1          self.trainingAccuracy = 0      #calculates the weighted sum of inputs      def weighted\_sum(self):          sum = 0          for i in range(len(self.x)) :              sum+= self.x[i]\*self.w[i]          #sum -= self.x[0]\*self.w[0]         # sum = round(sum,1)          return sum        # activation function      def step(self):          return 1 if self.weighted\_sum() > self.threshold else -1      # training method - changes weights until output is correct      def train(self):          while True:              output = self.step()              error = 0 if self.y == output else 1              #print(self.y, output, error)              if (error == 0):                  self.trainingAccuracy+=1                  return self.w              if(self.epochs >= 10000):                  return self.w              for i in range(len(self.x)):                  self.epochs+=1                  delta = self.alpha \*(self.y - output)\*self.x[i]                  self.w[i] = round(self.w[i] + (delta ),2)        # test perceptron by getting output from trained weights      def test(self, weights):          self.w = weights          output = self.step()          return output |

4-Pixel Test Class:

|  |
| --- |
| import Perceptron as P  import json  def partition(index):      numInputs = 5      avg\_weights = [0,0,0,0,0]      testDataIndex = index      with open ('data.json','r') as file:          data = json.load(file)          for i in range(16):              index = data['inputs'][i][2]              x  = data['inputs'][i][0]              y  = data['inputs'][i][1]              # if index matches the test index, create perceptron object to use for testing              if(index == testDataIndex):                  test = P.Perceptron(x,y,1)              # otherwise  create a training object and train it              else:                  #create perceptron object with inputs and desired output                  p = P.Perceptron(x,y,1)                  #get the weights from the perceptron                  final\_weights = p.train()                  for w in range(numInputs):                      avg\_weights[w] += final\_weights[w]      #get average weights from training      for i in range(numInputs):          avg\_weights[i] = round(avg\_weights[i]/numInputs,4)      # get output from testing with the trained weights      testOutput = test.test(avg\_weights)      # calculate test error      testError = 0 if test.y == testOutput else 1      print("Test Error: ",testError)      return testError  # run tests with partitions + calculate error  totalError = 0  for x in range(16):      xError = partition(x)      totalError += xError  print("Accuracy: ",16-totalError,"/",16, "=",round((16-totalError)/16,2)) |

CMYK Test Class:

|  |
| --- |
| import Perceptron as P  def train(n):      numInputs = 5      trainingData = generateData(n)      avg\_weights = [0,0,0,0,0]      trainingAccuracy=0      for sample in trainingData:          p = P.Perceptron(sample[0],sample[1],240)          final\_weights = p.train()          trainingAccuracy+= p.trainingAccuracy          for w in range(5):              avg\_weights[w] += final\_weights[w]      #get average weights      for i in range(numInputs):          avg\_weights[i] = round(avg\_weights[i]/numInputs,4)      return avg\_weights, trainingAccuracy  trainingSamples = 400  trained\_weights, trainingAccuracy = train(trainingSamples)  print("Training Accuracy:",trainingAccuracy,"/",trainingSamples)  def test(n,w):      testData =generateData(n)      totalError = 0      for sample in testData:          p = P.Perceptron(sample[0],sample[1],240)          # get output from testing with the trained weights          testOutput = p.test(w)          # calculate test error          testError = 1 if p.y == testOutput else -1          # calculate error          totalError += testError      print("Test Accuracy: ",n-totalError,"/",n, "=",round((n-totalError)/n,2))    test(100,trained\_weights) |

## Outputs:

A screen shot of a computer error

Description automatically generated

Cmyk output:

A black background with white text

Description automatically generated