Assignment 4 write-up

CISC352 Artificial Intelligence

April 4th 2017

Anna Ilina (10150979)

Jake Pittis (10148335)

Problem 1. Artificial Neural Networks: ORPerceptron

Four-Input OR-Training Using a Single-Layer Perceptron Network.

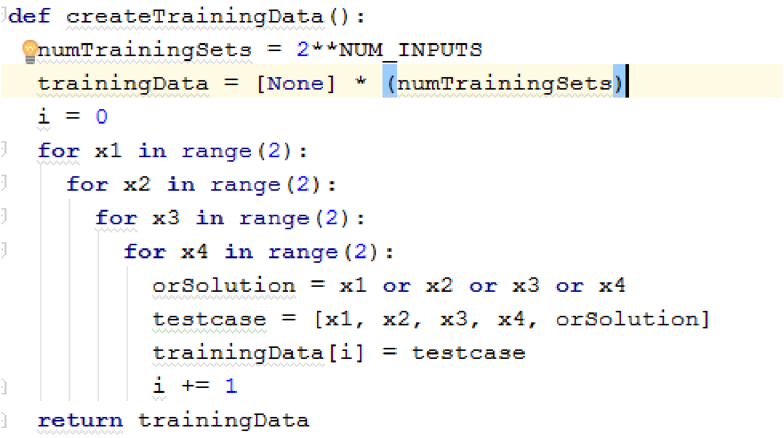
This problem was solved in Python 2.7, using the Random library.

The solution to this problem was pair-programmed by Jake and Anna.

Creating the training dataset

To create training data for the perceptron, I create an array of all the possible input sets to the 4-input OR function in the *createTrainingData()* method, and stored the OR target solution, in the format

[x1, x2, x3, x4, ORsolution]



```
def createTrainingData():
  numTrainingSets = 2**NUM_INPUTS
  trainingData = [None] * (numTrainingSets)
  i = 0
  for x1 in range(2):
    for x2 in range(2):
      for x3 in range(2):
        for x4 in range(2):
          orSolution = x1 or x2 or x3 or x4
          testcase = [x1, x2, x3, x4, orSolution]
          trainingData[i] = testcase
          i += 1
  return trainingData
```

Structure of the perceptron training function

Next, the perceptron was trained in the *trainPerceptron()* method. For training, 4 weights were used, corresponding to the 4 inputs. The weights were initialized as random values between -1 to 1 using the random.uniform() function. The program repeatedly loops through each of the 16 training sets, adjusting weights according to a step activation function when an input set is incorrectly classified, until it goes through a loop where all the training sets are classified correctly; at this point the perceptron is fully "trained" and the values of the weights are returned.

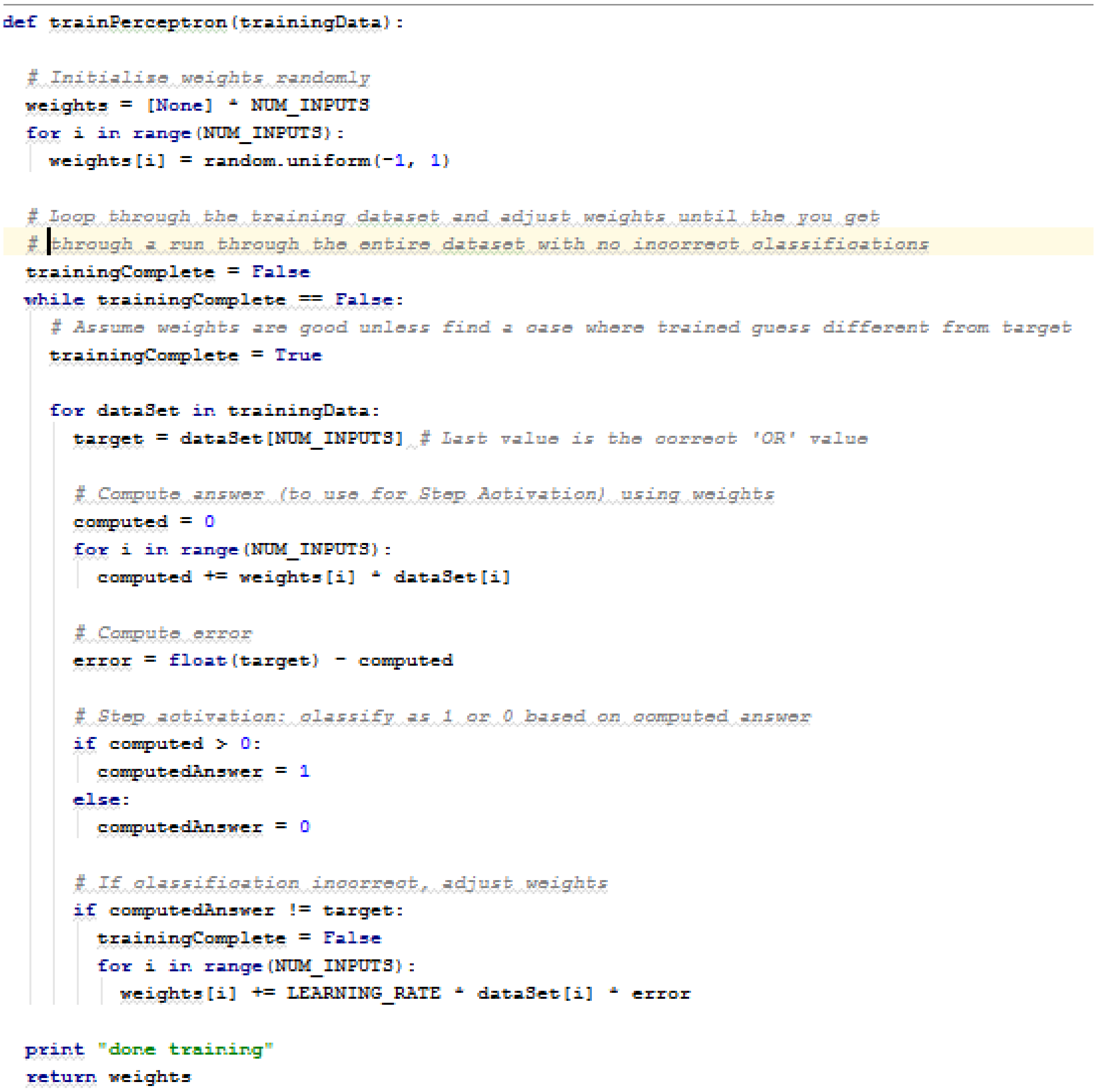
Step activation function

To train our perceptron, we used a step activation function with a threshold of t=0 and a learning rate of a=0.1. For inputs x_i and weights w_i, if the value $\Sigma(x_i * w_i)$ was above the threshold, the input set was classified as '1'. Otherwise, the input set was classified as '0'. If the input set was incorrectly classified, the weights were adjusted according to the following formula, where e is the error value (e = expected classification

$$w_i \leftarrow w_i + (a \times x_i \times e), ww$$

where e error from the expected classification (0 or 1), obtained by the forumula:

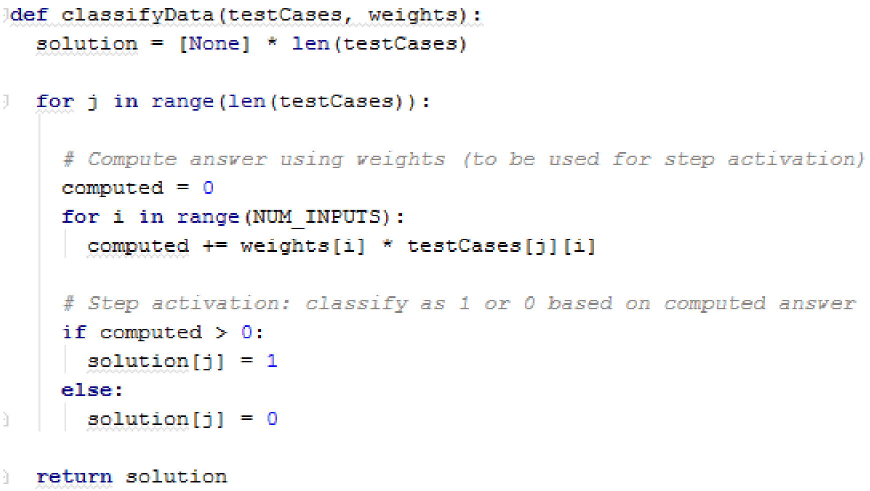
$$e = (\text{expected classification}) - \Sigma(x_i * w_i)$$



```
def trainPerceptron(trainingData):

  # Initialise weights randomly
  weights = [None] * NUM_INPUTS
  for i in range(NUM_INPUTS):
    weights[i] = random.uniform(-1, 1)

  # Loop through the training dataset and adjust weights until the you get
  # through a run through the entire dataset with no incorrect classifications
  trainingComplete = False
  while trainingComplete == False:
    # Assume weights are good unless find a case where trained guess different from target
    trainingComplete = True

    for dataSet in trainingData:
      target = dataSet[NUM_INPUTS] # Last value is the correct 'OR' value

      # Compute answer (to use for Step Activation) using weights
      computed = 0
      for i in range(NUM_INPUTS):
        computed += weights[i] * dataSet[i]

      # Compute error
      error = float(target) - computed

      # Step activation: classify as 1 or 0 based on computed answer
      if computed > 0:
        computedAnswer = 1
      else:
        computedAnswer = 0

      # If classification incorrect, adjust weights
      if computedAnswer != target:
        trainingComplete = False
        for i in range(NUM_INPUTS):
          weights[i] += LEARNING_RATE * dataSet[i] * error

  print "done training"
  return weights
```

Using trained perceptron to classify data

The weighs from the trained perceptron are used to classify input data using a step activation function in the *classifyData()* method.



```python
def classifyData(testCases, weights):
  solution = [None] * len(testCases)

  for j in range(len(testCases)):

    # Compute answer using weights (to be used for step activation)
    computed = 0
    for i in range(NUM_INPUTS):
      computed += weights[i] * testCases[j][i]

    # Step activation: classify as 1 or 0 based on computed answer
    if computed > 0:
      solution[j] = 1
    else:
      solution[j] = 0

  return solution
```