Assignment 3 Introduction to Deep Learning – CAP 4613

Due: Wednesday Feb 24

Instructions:

• The entire assignment has to be submitted either "electronically on Canvas" or on "paper and should be turned in at the beginning of the class" before the deadline.

• Each assignment should have the following information on the first page: assignment number, student name and znumber, and a shareable link to the final version of your Python code in Colab.

To get a shareable link, in your Colab notebook click 'Share' on the upper right corner, then click 'Get shareable link' and copy the link.

• The Python submission should include the codes and the generated outputs.

To generate the PDF submission file: in Colab, go to 'File'=>'Print' then change the 'Destination' to 'Save as PDF' and save.

• **Only one file** has to be submitted electronically. Combine the handwritten and Python parts before submission.

You can upload the pictures of your handwritten answers to the drive, and then import and show them in Colab notebook using matplotlib library.

• Filename for electronic submission: Student_Name_Assignmentxx.pdf or doc.

Problem 1) Perceptron learning in Python:

a) Create class NeuralNetwork(): that creates a single neuron, train it, and test it. This class should have the following function:
   i. def __init__(self): that initializes a 3x1 weight vector randomly and initializes the learning rate to 1.
   ii. def hard_limiter(self, x): that performs the hard-limiter activation on the nx1 vector x.
   iii. def forward_propagation(self, inputs): that performs the forward propagation by multiplying the inputs by the neuron weights and passing the output through the hard_limiter activation function.
   iv. def train(self, inputs, labels, num_train_iterations=10): that performs the perceptron learning rule for num_train_iterations times using the inputs and labels.
   v. def pred(self,inputs): classifies the inputs to either class 0 or 1 by multiplying them by the neuron weights, passing the output through the hard_limiter activation function and thresholding.

b) Use the perceptron learning rule to train a single neuron on the data points given in problem 3 of Assignment 2:
   i. Create an np array of the shape of 6x2 that contains the inputs, and another array with the shape of 6x1 that contains the labels.
   ii. Add the bias to the input array to have a 6x3 shape.
   iii. Create the network with one perceptron using the class NeuralNetwork(), then train it using train(inputs, labels,100) function.
c) Plot the given data points with two different markers for each group.
d) Using the trained perceptron weight, plot the classifier line in the same plot in (c).
e) Using the trained perceptron, classify the test data samples given in the table below by calling the pred() function.

input x_1	input x_2	desired label	predict the label
2	0	1	
2	1	0	
0	0	1	
-2	0	0	

Problem 2) Gradient descent learning: Consider the following set of data points:

input x_1	input x_2	desired label
1	1	1
1	0	1
0	1	0
-1	-1	0
-1	0	0
-1	1	0

As the above table shows, the data points are categorized (labeled) in two groups specified by the labels "1" and "0".

a) Use the gradient descent learning algorithm to train a single neuron on the data samples given in the table above.
   Repeat the gradient descent algorithm for only 3 iterations.
   Assume the learning rate is 0.1 with initial weight vector of (1 1 1).
a) Draw the average square error all the three iterations. Is the error decreasing? Explain your observation.
b) Draw the schematics of the trained classifier after the third iteration.
c) Provide the equation of the trained model at every iteration. Hint: the question of the three classifier lines.
d) Plot the given data points with two different markers for each group.
e) Plot the classifier line of the 3rd iteration to the plot in (d). Label each classifier.
f) Calculate the accuracy, sensitivity, and specificity of the 3rd classifier.

Problem 3) Gradient descent learning in Python:

f) Create class NeuralNetwork(): that creates a single neuron with a linear activation, train it using gradient descent learning. This class should have the following function:
   i. def __init__(self, learning_r): that initializes a 3x1 weight vector randomly and initializes the learning rate to learning_r. Also, it creates a history variable that saves the weights and the training cost after each epoch (i.e., iteration).
   ii. def forward_propagation(self, inputs): that performs the forward propagation by multiplying the inputs by the neuron weights and then generating the output.
   iii. def train(self, inputs_train, labels_train, num_train_epochs=10): that performs the gradient descent learning rule for num_train_epochs times using the inputs and labels. This function also saves the weights and costs at every epoch to the history variable.

g) Use the gradient descent rule to train a single neuron on the datapoints, given below:
   i. Create an np array of a shape 10x2 that contains the inputs, and another array with a shape 10x1 that contains the labels.
   ii. Add the bias to the inputs array to have a 10x3 shape.
   iii. Create the network with one neuron using the class NeuralNetwork() with learning rate of 1 then train it using train(inputs, labels, 50) function.

input		desired
x_1	x_2	label
1	1	1
1	0	1
0	1	-1
-1	-1	-1
0.5	3	1
0.7	2	1
-1	0	-1
-1	1	-1
2	0	1
-2	-1	-1

h) Plot the given data points with two different markers for each group.
i) Use the trained weights at each epoch and plot the classifier lines of every 5 epochs (i.e., 1,5,10, …, 50) to the plot in (c).
j) Use the trained weights and plot the final classifier line to the plot in (c).
k) Plot the training cost (i.e., the learning curve) for all the epochs.
l) Repeat step (b) with the learning rates of 0.5 and 0.05. Create a subplot with the learning curves of learning rates 1, 0.5, and 0.05. Add titles.
m) What behavior do you observe from the learning curves with the different learning rates? Explain your observations. Which learning rate is more suitable? Explain.