# Homework#1

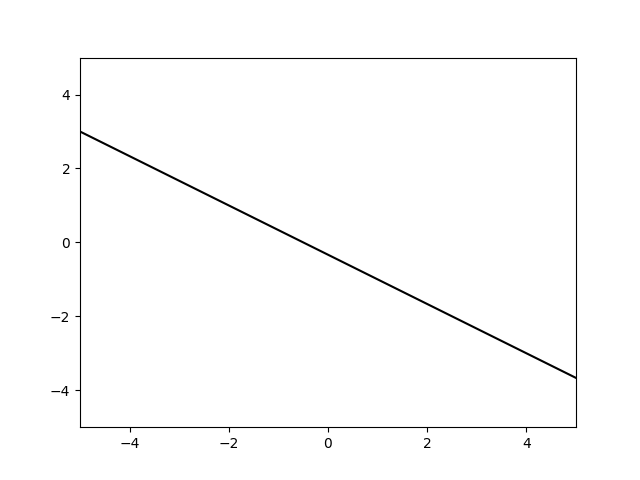
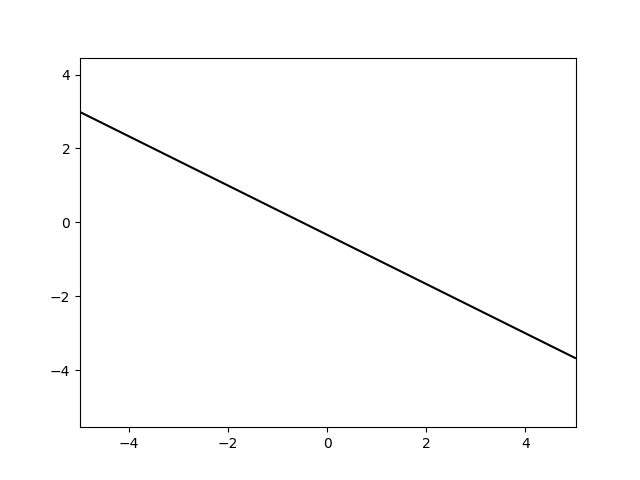
## Problem 1.2:

|  |
| --- |
| There are two solution space: +1 and -1  Hence the equation w0+w1x1+w2x2 can have two possible solutions as:  w0+w1x1+w2x2 = +1 and w0+w1x1+w2x2 =-1  adding both equations gives us:  2(w0+w1x1+w2x2) =0, hence w0+w1x1+w2x2=0  We can re-write this as: x2= (-w1/w2)\*x1 + (-w0/w2)  This equation is similar to equation of a line: y=mx+b  Hence, we can conclude that the perceptron function f, represents a line for two-dimensional space (for two features) and thus +1 and -1 regions can be separated by a line with the right weight vector and classifiable features. |

* 1. Slope(a) = -(w1/w2) in W vector [w0,w1,w2] for line ax+b.
  2. Intercept(b) = -w0/w2 in W vector [w0,w1,w2] for line ax+b.

1. Plotting using the following code for W=[1,2,3] and W=-[1,2,3]. Because both have same slope and intercepts and the sign is different, we get similar lines with opposite slopes:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  W=np.array([1,2,3]) # or W=-W  a, b = -(W[0]/W[2])/(W[0]/W[1]), -W[0]/W[2]  plt.xlim(-5,5)  plt.ylim(-5,5)  l = np.linspace(-5,5)  plt.plot(l, a\*l+b, 'k-')  plt.show() |

* 1.  W=[1,2,3]. Positiive slope
  2. W=-[1,2,3], negative slope

## Exercise 1.4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D=2   |  |  |  | | --- | --- | --- | | X1 | X2 | yn | | 1 | 2 | +1 | | 3 | 4 | +1 |   Let the weight vector of a random target functions that classifies the dataset be [2,2,2]  We get a line equations as: x2=-x1-1  Solving the equation for f(x) = sign(wT.x) to generate an output  Using d0(1,2) as x : [1,1,2]  F(x) = sign(w\_trans\*x) = +1 = yn |

We generate the dataset using :

|  |
| --- |
| def generate\_dataset(self, N):  dataSet = []  for i in range(N):  #generate x1,x2 for ith row in data set  feature1,feature2 = [random.uniform(-5, 5) for i in range(2)]    #feature vector for ith row [x0,x1,x2]  featureVector = np.array([1,feature1,feature2])    #solution space for xi, s=sign(WT.X)  output = int(np.sign(self.targetWeight.T.dot(featureVector)))    dataSet.append((featureVector, output))  return dataSet |

Following is the resulting data set:

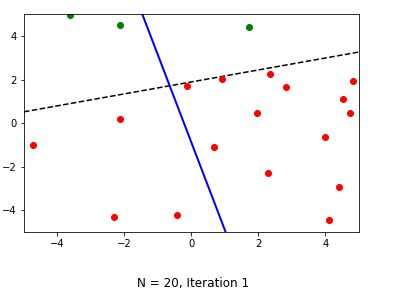
|  |
| --- |
| **X1 X2 Yn**  2.34009606418 2.26568724668 :: -1  -0.132746432482 1.70486548125 :: -1  -3.61180422644 4.97686167045 :: 1  0.915504922401 2.00628549029 :: -1  -2.12655357331 0.191197826504 :: -1  -2.11854016978 4.52762849069 :: 1  2.82647318539 1.65008683904 :: -1  4.12228055613 -4.45289193678 :: -1  1.948037446 0.440837408103 :: -1  1.72650238426 4.41866892447 :: 1  4.40327162385 -2.93396759048 :: -1  4.83644726508 1.9177160881 :: -1  -0.439618365372 -4.23542970363 :: -1  3.99852631175 -0.636668854647:: -1  -4.71827545478 -1.00053868146 :: -1  4.52498675027 1.12432333735 :: -1  4.74934596659 0.469247514604 :: -1  0.677028668155 -1.07723083695 :: -1  2.28301944615 -2.27911136281 :: -1  -2.29693843215 -4.34143035292 :: -1 |

Following code is used to compute and plot PLA:

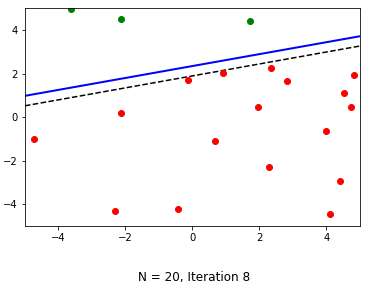
|  |
| --- |
| 1. **import** numpy as np 2. **import** random 3. **import** matplotlib.pyplot as plt 5. **class** PLA\_14: 6. **def** \_\_init\_\_(self, N, err=0): 7. # Random linearly separated data 8. xA,yA,xB,yB = [random.uniform(-5, 5) **for** i **in** range(4)] 10. #V is the weight vector with values[w0,w1,w2] generated in a random uniform manner. 11. self.targetWeight = np.array([xB\*yA-xA\*yB, yB-yA, xA-xB]) 13. #this is the data set of N rows 14. self.dataSet = self.generate\_dataset(N) 15. self.classificationErr=err 17. **def** generate\_dataset(self, N): 18. dataSet = [] 19. **for** i **in** range(N): 20. #generate x1,x2 for ith row in data set 21. feature1,feature2 = [random.uniform(-5, 5) **for** i **in** range(2)] 23. #feature vector for ith row [x0,x1,x2] 24. featureVector = np.array([1,feature1,feature2]) 26. #solution space for xi, s=sign(WT.X) 27. output = int(np.sign(self.targetWeight.T.dot(featureVector))) 29. dataSet.append((featureVector, output)) 31. **return** dataSet 33. **def** plot\_graph(self, currHypoWeight=[]): 34. plt.xlim(-5,5) 35. plt.ylim(-5,5) 36. weightVector = self.targetWeight 37. slope, intercept = -(weightVector[1]/weightVector[2]), -weightVector[0]/weightVector[2] # a=slope, b= coeff if line 38. l = np.linspace(-5,5) 39. plt.plot(l, slope\*l+intercept, 'k--') # plot a line al+b 40. cols = {1: 'g', -1: 'r'} 41. **for** feature,output **in** self.dataSet: 42. plt.plot(feature[1], feature[2], cols[output]+'o') 43. **if** len(currHypoWeight)!=0: 44. hypothesis\_slope  = -currHypoWeight[1]/currHypoWeight[2] 45. hypothesis\_intercept = -currHypoWeight[0]/currHypoWeight[2] 46. plt.plot(l, hypothesis\_slope\*l+hypothesis\_intercept, 'b-', lw=2) 48. plt.show() 50. **def** classification\_error(self, currWeights): 51. # Error defined as fraction of misclassified points 52. datSet = self.dataSet 53. M = len(datSet) 54. datSet\_mispts = 0 55. **for** features,output **in** datSet: 56. **if** int(np.sign(currWeights.T.dot(features))) != output: 57. datSet\_mispts += 1 58. error = datSet\_mispts / float(M) 59. **return** error 61. **def** rand\_missclassfied\_pt(self, currHypoWeight): 62. # Choose a random point among the misclassified 63. datSet = self.dataSet 64. mispts = [] 65. **for** features,output **in** datSet: 66. **if** int(np.sign(currHypoWeight.T.dot(features))) != output: # check if h(x) != f(x) 67. mispts.append((features, output)) 68. **return** mispts[random.randrange(0,len(mispts))] # Return the missclassified point 70. **def** pla(self): 72. currHypoWeight = np.zeros(3) # weight vector of current hypotheses 73. N = len(self.dataSet) 74. iterations = 0 75. # Iterate until all points are correctly classified 77. **while** self.classification\_error(currHypoWeight) > self.classificationErr: 78. iterations += 1 79. # Pick random misclassified point 80. missclass\_features, missclass\_output = self.rand\_missclassfied\_pt(currHypoWeight) 81. # Update weights 82. nextHypoWeight = currHypoWeight + missclass\_features\*missclass\_output 83. currHypoWeight=nextHypoWeight 84. self.plot\_graph(currHypoWeight) 85. plt.title('N = %s, Iteration %s\n' 86. % (str(N),str(iterations))) 88. plt.show() 90. self.finalWeight = currHypoWeight 91. self.iterations = iterations 93. **def** print\_dataset(self): 94. **for** feature,output **in** self.dataSet: 95. **print**(feature[1],"  ",feature[2]," :: ",output) 97. **def** print\_final\_weights(self): 98. **print**("Weight vector of learned hypothesis function g is \n",self.finalWeight) 100. **def** print\_iterations(self): 101. **print**("Iterations it took to converge towards the target function:   \n",self.iterations)     107. p = PLA\_14(20,0) # N,error value -- to missclassify 3 points out of N, then pass 3/N as err value. 108. p.plot\_graph()     #plot target function shown by black line and random points which are classified into +1(blue) and -1(red) 110. p.pla() #calling perceptron model (blue line is hypothesis and black is target function) 111. p.print\_dataset() # print dataset that is generated. 112. p.print\_final\_weights() # this is the learned weights after running PLA algorithm. 113. p.print\_iterations() |

## Problem 1.4

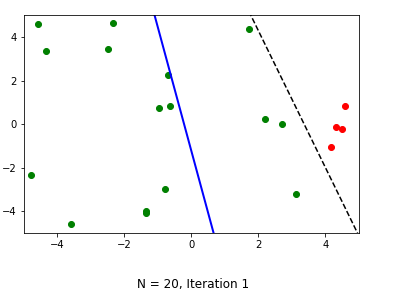
1. For the above dataset, target function f is shown in black and the initial hypothesis function h is shown in blue. Iteration =1. PLA starts from this point and iterates 8 times to converge.

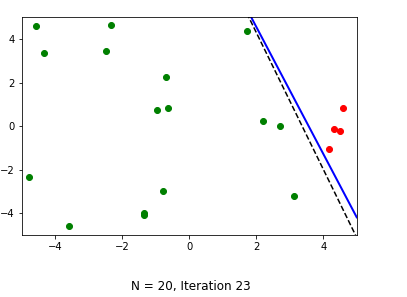


1. PLA converges at **iteration** **8** for this set of data points. H(x) ~= f(x). The final hypothesis function g is very close to the target function f.

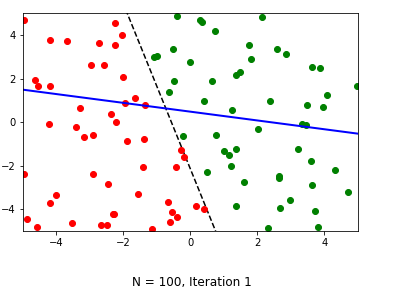


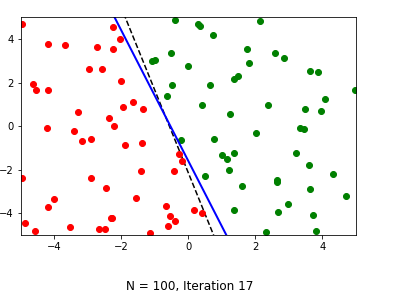
1. Following is based on another run of PLA over another set of random data of size 20. Black line represents target function f and blue line represents hypothesis function h. For this set of Data, PLA takes 23 iteration to converge (correctly classifies all data points). The final hypothesis function g and target function f are very close.





1. Following is based on another run of PLA over another set of random data of size **100**. Black line represents target function f and blue line represents hypothesis function h. For this set of Data, PLA takes **17 iterations** to converge (correctly classifies all data points). The final hypothesis function g and target function f are very close.





Final weights that lead to convergence:

|  |
| --- |
| [ 5, 9.32552898, 3.10890916] |

Data set for above plot:

|  |
| --- |
| -0.231812655307 -0.617539850761 :: 1  -2.24236339149 4.57222279608 :: -1  -2.89586898189 -0.613657201218 :: -1  -0.176020076628 -1.62156545344 :: -1  0.161274542573 -3.87569958283 :: -1  1.36231897995 2.14543820346 :: 1  1.35994513576 -3.85580755072 :: 1  -0.484618352487 1.9099047082 :: 1  -2.21669310412 0.0186142966951 :: -1  -4.96757974832 -2.40468996106 :: -1  -0.535122069018 -4.14554274369 :: -1  0.497992657697 -2.2863259888 :: 1  -1.07452099302 3.01143758288 :: 1  -1.39892823718 -0.769360334086 :: -1  -0.293634700831 -1.29640721171 :: -1  -2.28729563842 -4.21794380685 :: -1  0.29855746351 4.71491201576 :: 1  1.59405627818 -2.7698516728 :: 1  -3.67033001859 3.73268689629 :: -1  2.02793658393 -0.31753119442 :: 1  -4.88197049573 -4.44201063799 :: -1  2.65631411547 -2.59298661217 :: 1  3.46724098507 -0.148918985664 :: 1  0.338286722676 4.58567976493 :: 1  3.47803086143 0.792700666821 :: 1  -2.38210845129 0.371115033685 :: -1  1.82037508895 2.8945132247 :: 1  -4.55831464764 1.67860772211 :: -1  -4.64102399629 1.91784569869 :: -1  -4.0244414687 -3.35617743381 :: -1  4.30952005135 -2.19386670167 :: 1  0.775896565874 -0.597996213906 :: 1  -2.48094641099 -4.72157183686 :: -1  -1.90491385858 -0.863072244005 :: -1  3.88661147694 2.48850581605 :: 1  -4.58225185211 -4.83598151382 :: -1  -2.00932096758 2.06996478922 :: -1  -4.19729209306 1.66081577413 :: -1  2.85747983666 3.13162899532 :: 1  3.95367181388 0.695490423605 :: 1  -0.0141589300726 2.77454253267 :: 1  1.37535453915 -1.23770120304 :: 1  -2.30457971696 -4.22402086763 :: -1  -2.25080508354 3.52849191562 :: -1  0.995828252268 -1.30917807148 :: 1  2.9906553607 -3.59831771468 :: 1  3.33036568614 -0.107688704201 :: 1  -4.96530684183 4.70088756356 :: -1  -3.3016274137 0.64897625619 :: -1  -0.526221125514 3.36113659237 :: 1  -4.19719278818 -3.72652967098 :: -1  -1.3631256598 0.765619600016 :: -1  -2.58863254836 2.61549755886 :: -1  -4.23533380144 -0.0841290449379 :: -1  -2.45660531534 -2.86887453206 :: -1  2.32502805362 -4.86186389813 :: 1  -1.6512352794 1.11194841037 :: -1  1.74902614251 3.5505520522 :: 1  -2.89500154914 -2.4076930752 :: -1  -1.96557858875 0.856348854186 :: -1  1.22540485974 -2.03660559402 :: 1  4.08558268122 1.25070603234 :: 1  -0.629126805939 1.40590403996 :: 1  0.750045956631 4.16814164919 :: 1  -0.604834656638 -4.59891357159 :: -1  3.61823244843 -1.80585736916 :: 1  -3.54529994516 -4.62389944456 :: -1  1.47854795968 2.30134336678 :: 1  -4.1849373596 3.77066071634 :: -1  -3.41623446822 -0.243115696595 :: -1  -3.16662625125 -0.704437635315 :: -1  0.652732303172 1.87176268774 :: 1  1.25445080472 0.550490623634 :: 1  -2.03030576259 4.00453208667 :: -1  -2.95982384179 2.64413049822 :: -1  3.22684897989 -1.24598710513 :: 1  0.399601051359 -4.01919091759 :: -1  3.80874297956 -4.80528960352 :: 1  2.57596177025 3.37906622476 :: 1  -0.412673667947 -4.36298702004 :: -1  -2.66485278585 -4.7394854029 :: -1  -0.410445428031 4.87297311156 :: 1  2.14808773181 4.82761175514 :: 1  3.62574133359 2.53753866098 :: 1  2.66596614345 -3.9567909951 :: 1  1.15870600704 -1.51290004639 :: 1  -1.57281755965 -3.30656140662 :: -1  -1.15803426261 -4.93819332808 :: -1  0.40467899544 0.953973911248 :: 1  3.71225634496 -4.10971216608 :: 1  3.63943243316 -2.88394948016 :: 1  4.97654218056 1.63711765953 :: 1  -2.73068094721 3.65572214117 :: -1  4.71128744335 -3.23507555616 :: 1  2.65375618741 -2.48014475423 :: 1  -0.99657909021 3.02832612101 :: 1  -0.421764208134 -2.08677863849 :: -1  -0.653164362449 -3.65686767996 :: -1  2.36766978834 0.951885449681 :: 1  -1.42911941941 -2.05150034465 :: -1 |