CS 4340 - Project PLA

Austin Hester

September 14, 2017

Introduction

The Perceptron Learning Algorithm (PLA), is a classic machine learning algorithm. It works by iteratively changing the weights of every input category, x_i. It creates a plane (in our case, a line) using these weights, and checks if there are any misclassified points.

If any misclassified points are found, it edits the weights based on a single, randomly chosen misclassified point. I decided to pick a random misclassified point instead of the first misclassified point because an outying point may hinder its execution. The algorithm runs until it finds no misclassified points.

I have chosen to generate new training points for each run. This decision was made to test the PLA on various inputs. Testing data is also newly generated for every test done. The figure below represents a final iteration of training.

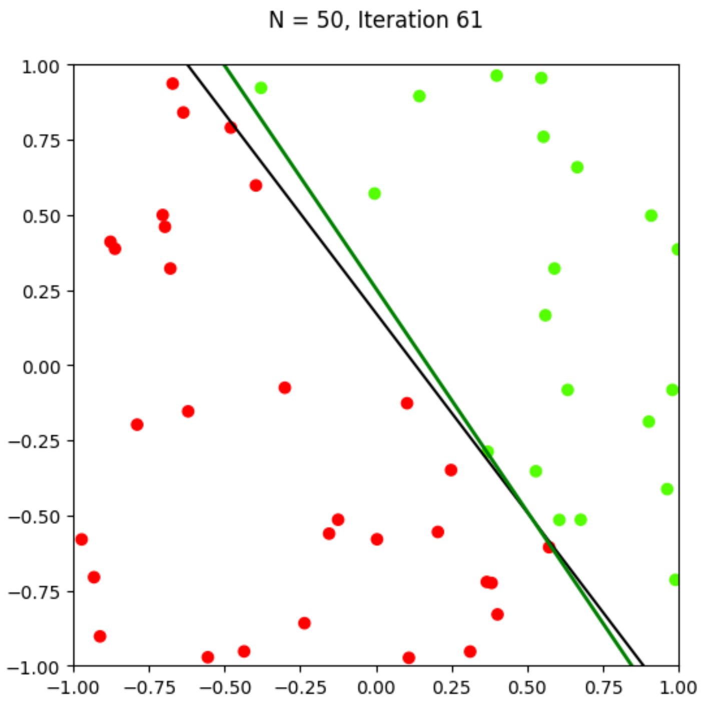
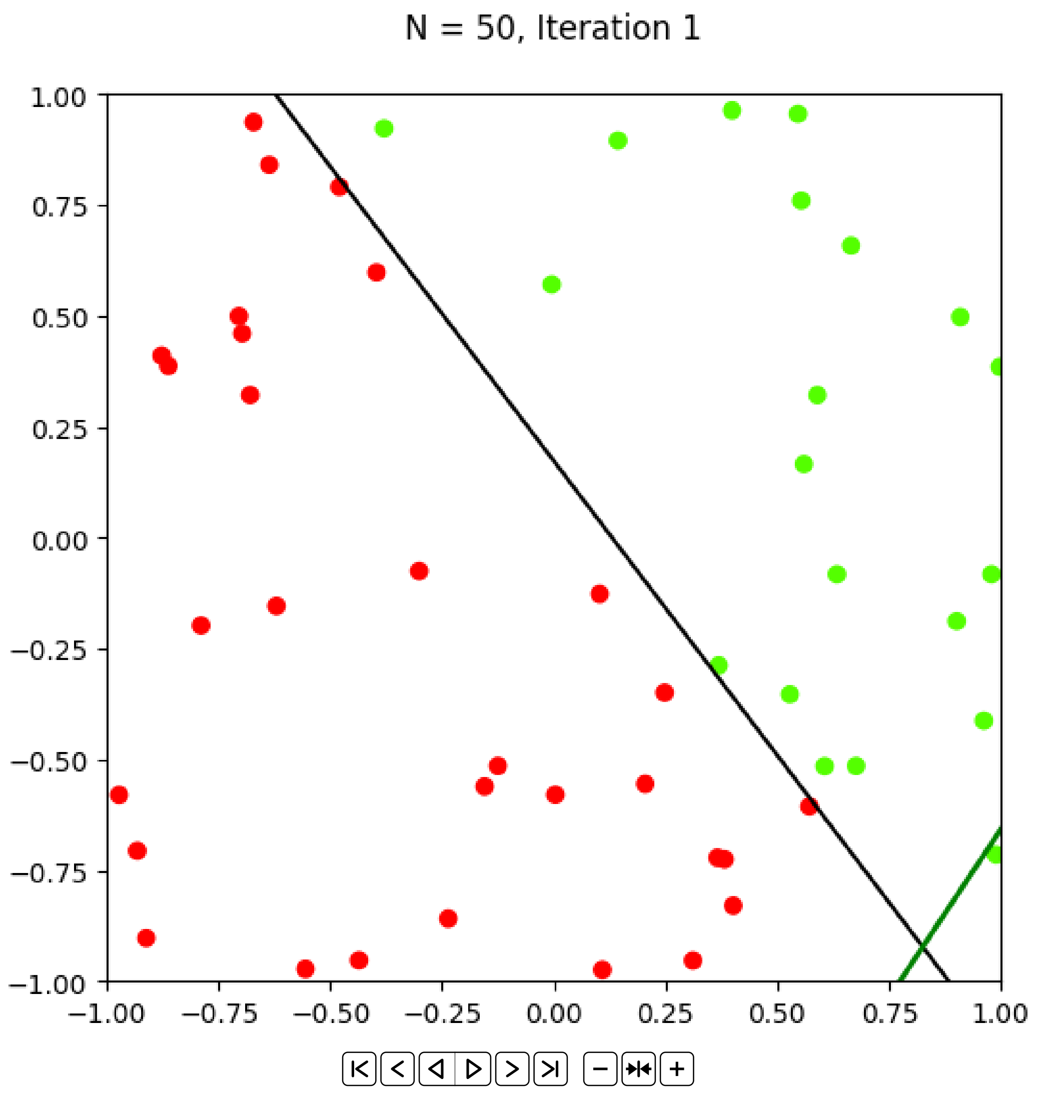


Figure 1: Final Iteration of PLA Training. *Unknown* function, f, shown in black. PLA's chosen hypothesis, g, shown in green.

Animated GIF of PLA Training



The Code

```
# Austin Hester
# 09/13/2017
# PLA Python Implementation
# Trains with 50 linearly seperable points
# Tests against 30
import numpy as np
import random
import matplotlib.pyplot as plt

class Perceptron:
    def __init__(self, N):
        x1, y1, x2, y2 = [random.uniform(-1, 1) for i in range(4)]
        # for generating linearly seperable data (V)
        self.V = np.array([x2*y1-x1*y2, y2-y1, x1-x2])
        self.X = self.generatePoints(N)
        self.iterations = 0

    def generatePoints(self, N):
        X = []
        for i in range(N):
            x1, x2 = [random.uniform(-1, 1) for i in range(2)]
            x_ = np.array([1, x1, x2])
            # classify based on V, our PLA does not know this line
            s = int(np.sign(self.V.T.dot(x_)))
            x_ = np.append(x_, [s])
            X.append(x_)
        return np.array(X)

    def plot(self, testPts=None, w_=None, save=False):
        fig = plt.figure(figsize=(6,6))
        plt.xlim(-1,1)
        plt.ylim(-1,1)
        plt.title('N = %s, Iteration %s\n' % (str(len(self.X)),str(self.
            iterations)))
        # draw line pla is searching for
        V = self.V
        a, b = -V[1]/V[2], -V[0]/V[2]
        l = np.linspace(-1,1)
        plt.plot(l, a*l+b, 'k-')
        ax = fig.add_subplot(1,1,1)
        ax.scatter(self.X[:,1:2], self.X[:,2:3], c=self.X[:,3:4], cmap='prism')
        if (w_ is not None and w_[2] != 0):
            # draw training line
            aa, bb = -w_[1]/w_[2], -w_[0]/w_[2]
            plt.plot(l, aa*l+bb, 'g-', lw=2)
        if (testPts is not None):
            # draw test points
            ax.scatter(testPts[:,1:2], testPts[:,2:3], c=testPts[:,3:4], cmap='
                cool')
        if save:
            plt.savefig('.\gifs\p_N%s' % (str(len(self.X))), dpi=100, bbox_inches
                ='tight')
        else:
            plt.show()

    # returns percentage of missed points
    def classifyError(self, w_, pts=None):
        if pts is None:
            pts = self.X[:,:3]
            S = self.X[:,3:4]
        else:
            S = pts[:,3:4]
            pts = pts[:,:3]
        M = len(pts)
        n_mispts = 0
        for x_, s in zip(pts, S):
            if int(np.sign(w_.T.dot(x_))) != s:
                n_mispts += 1
        print("Missed points: %d" % (n_mispts))
        print("w_: ", w_)
        err = n_mispts / float(M)
        return err

    # Pick a random misclassified pt (according to given w_)
    def pickMisclPoint(self, w_):
        pts = self.X[:,:3]
        S = self.X[:,3:4]
        mispts = []
        for x,d in zip(pts, S):
            if int(np.sign(w_.T.dot(x))) != d:
                mispts.append((x, d))
        return mispts[random.randrange(0, len(mispts))]

    # Run PLA on data contained in self.X
    def pla(self, c=0.01, save=False):
        X = self.X[:,:3]
        N = len(X)
        w_ = np.zeros(len(X[0]))
        it = 0
        print("Iteration %d: " % (it))
        # Run while there are misclassified points
        # or we get to 1,000 iterations
        while self.classifyError(w_) != 0 or it > 1000:
            it += 1
            print("Iteration %d: " % (it))
            # pick mispicked pt
            x, d = self.pickMisclPoint(w_)
            w_ += c*d*x
            if save:
                self.plot(vec=w_, save=True)
                plt.title('N = %s, Iteration %s\n' % (str(N),str(it)))
                plt.savefig('.\gifs\p_N%s_it%s' % (str(N),str(it)), dpi=100,
                    bbox_inches='tight')
        self.w = w_
        self.iterations = it

    # Test our test points using classifyError
    def checkTestPoints(self, testPts, w_):
        print("————————————————")
        print("Test Info")
        print("————————————————")
        return self.classifyError(w_, pts=testPts)


def testPLA(p, M):
    testPts = p.generatePoints(M)
    print("————————————————")
    print("Testing data:\n", testPts)
    print("————————————————")
    testError = p.checkTestPoints(testPts, p.w)
    print("Test Error:", round(testError * 100, 3), "%")
    print("————————————————")
    return testPts, testError

# initialize perceptron with 50 training points
train = 50
test = 30
save = False
p = Perceptron(train)
print("————————————————")
print("Training data:\n", p.X)
print("————————————————")
p.pla(save=save) # run pla, save=True to generate gif

testPts, testErr = testPLA(p, test)

if not save:
    p.plot(testPts=testPts, w_=p.w)
```

Notes

In order to test the PLA, **1)** fifty linearly seperable training points are generated at the *init* of the Perceptron class. These training points are generated using random.uniform(-1, 1). They are classified depending on our “unknown” function “V”, which is known to the point generator but not to the learning algorithm itself.

2) The test points are generated using the same algorithm, using the same “unknown” function to classify them. However, we will use our line generated by our Perceptron Learning algorithm to predict the classification of these points. The line we generate is built from the weights of each x_i, which we calculated during training.

3) The weights are initiated at zero, and the “step size,” $c = 0.01$.

6) In the given GIF, the PLA took 61 iterations to come up with a weight vector which properly classifies each point. **5)** Thus the PLA made 61 changes to the weight vector on this run.

7) The PLA in this case came out with 0.0% error on the training data as well as 0.0% error on the test data. On other runs with 50 training points, the weight vector commonly misclassifies one or two test points (3-7% error), but usually comes out with 0.0% error.

Example Run

Here is example output of a run which took only 11 iterations:

```
Training data:
 [[ 1.          0.77424746 -0.67735425 -1.        ]
 [ 1.          0.70243284  0.74123117  1.        ]
 [ 1.         -0.5135216  -0.50709209 -1.        ]
 [ 1.          0.48516912 -0.35136646 -1.        ]
 [ 1.         -0.68969807 -0.41902561 -1.        ]
 [ 1.         -0.95003182  0.63786985  1.        ]
 [ 1.          0.32586693  0.76960758  1.        ]
 [ 1.         -0.33664639  0.45631509  1.        ]
 [ 1.          0.20196235 -0.00205313 -1.        ]
 [ 1.          0.11494734 -0.6889786  -1.        ]
 [ 1.         -0.83852884 -0.91158334 -1.        ]
 [ 1.          0.36472927  0.23409315 -1.        ]
 [ 1.          0.0219022  -0.88920262 -1.        ]
 [ 1.          0.69709655 -0.72372811 -1.        ]
 [ 1.         -0.25887744  0.90189015  1.        ]
 [ 1.         -0.8943957  -0.48882512 -1.        ]
 [ 1.         -0.01583293 -0.35662745 -1.        ]
 [ 1.          0.39398236  0.13768795 -1.        ]
 [ 1.          0.33600582  0.01072832 -1.        ]
 [ 1.          0.16458495  0.30385782 -1.        ]
 [ 1.          0.22649929 -0.94639008 -1.        ]
 [ 1.          0.63694666  0.88084482  1.        ]
 [ 1.         -0.75826616 -0.49437639 -1.        ]
 [ 1.         -0.3083029   0.17414661 -1.        ]
 [ 1.         -0.1688848  -0.29456037 -1.        ]
 [ 1.          0.1873525  -0.14520642 -1.        ]
 [ 1.          0.26241554  0.26838309 -1.        ]
 [ 1.          0.69868185  0.59006842  1.        ]
 [ 1.         -0.03022481 -0.50473248 -1.        ]
 [ 1.         -0.88230351 -0.27371212 -1.        ]
 [ 1.          0.94223057  0.47595457  1.        ]
 [ 1.         -0.05403395  0.13518503 -1.        ]
 [ 1.          0.57126467  0.37647213 -1.        ]
 [ 1.          0.62088951  0.02757454 -1.        ]
 [ 1.          0.21383953 -0.27894893 -1.        ]
 [ 1.          0.51272036  0.87014766  1.        ]
 [ 1.          0.39548343 -0.31108663 -1.        ]
 [ 1.         -0.89106914  0.21489208 -1.        ]
 [ 1.          0.28803738  0.12356214 -1.        ]
 [ 1.         -0.78638381  0.59333724  1.        ]
 [ 1.         -0.20826485 -0.38793225 -1.        ]
 [ 1.         -0.02779911 -0.07494006 -1.        ]
 [ 1.          0.268374    0.22031122 -1.        ]
 [ 1.          0.64183295 -0.35348096 -1.        ]
 [ 1.          0.04516031  0.09253827 -1.        ]
 [ 1.         -0.9206282  -0.93456575 -1.        ]
 [ 1.          0.00241242 -0.30603518 -1.        ]
 [ 1.         -0.60449552  0.80100676  1.        ]
 [ 1.         -0.64633453  0.67516056  1.        ]
 [ 1.         -0.33852799  0.1008104  -1.        ]]
------------------------
Iteration 0:
Missed points: 50
w_:  [ 0.  0.  0.]
Iteration 1:
Missed points: 12
w_:  [-0.01        -0.00571265 -0.00376472]
Iteration 2:
Missed points: 20
w_:  [ 0.          -0.01357648  0.00216865]
Iteration 3:
Missed points: 35
w_:  [ 0.01        -0.00720702  0.0109771 ]
Iteration 4:
Missed points: 14
w_:  [ 0.          -0.00031004  0.01516736]
Iteration 5:
Missed points: 5
w_:  [-0.01        -0.00076164  0.01424197]
Iteration 6:
Missed points: 11
w_:  [ 0.          -0.01026196  0.02062067]
Iteration 7:
Missed points: 7
w_:  [-0.01        -0.01288611  0.01793684]
Iteration 8:
```

```
Missed points: 12
w_:  [ 0.          -0.00586179  0.02534915]
Iteration 9:
Missed points: 2
w_:  [-0.01        -0.00631339  0.02442377]
Iteration 10:
Missed points: 15
w_:  [ 0.           0.00310892  0.02918332]
Iteration 11:
Missed points: 0
w_:  [-0.01        -0.00083091  0.02780644]
-------------------------
Testing data:
 [[ 1.          -0.12599074  0.43038419  1.        ]
 [ 1.          0.20063998  0.31409969 -1.        ]
 [ 1.          0.35033788  0.58026622  1.        ]
 [ 1.          0.88013457 -0.95826867 -1.        ]
 [ 1.          0.59576249  0.17969005 -1.        ]
 [ 1.         -0.69497823  0.52857416  1.        ]
 [ 1.          0.61930789 -0.77867782 -1.        ]
 [ 1.         -0.81450926 -0.42581748 -1.        ]
 [ 1.         -0.05551606 -0.90144657 -1.        ]
 [ 1.          0.17073374 -0.35470633 -1.        ]
 [ 1.         -0.43192139 -0.38487801 -1.        ]
 [ 1.         -0.85636962 -0.73846554 -1.        ]
 [ 1.         -0.45945634 -0.27589654 -1.        ]
 [ 1.         -0.97264951  0.51724398  1.        ]
 [ 1.         -0.85669043 -0.43163207 -1.        ]
 [ 1.          0.28624344 -0.72241195 -1.        ]
 [ 1.          0.52276186 -0.45272691 -1.        ]
 [ 1.         -0.28597661 -0.08043544 -1.        ]
 [ 1.          0.91858526  0.39639456 -1.        ]
 [ 1.          0.9388579  -0.62246098 -1.        ]
 [ 1.         -0.61552446  0.74270188  1.        ]
 [ 1.         -0.73185148 -0.31567829 -1.        ]
 [ 1.         -0.84723934  0.86464977  1.        ]
 [ 1.         -0.88313756 -0.57560282 -1.        ]
 [ 1.          0.79714716 -0.97272488 -1.        ]
 [ 1.         -0.52573813 -0.96214108 -1.        ]
 [ 1.          0.06266416  0.83974583  1.        ]
 [ 1.          0.80285758 -0.34306834 -1.        ]
 [ 1.          0.07441788  0.65933061  1.        ]
 [ 1.         -0.49732753 -0.5687117  -1.        ]]
-------------------------
Test Info
-------------------------
Missed points: 1
w_:  [-0.01        -0.00083091  0.02780644]
Test Error: 3.333 %
```

4) Our final equation of the line is: $a * l + b$,
where $a = -w\_[1]/w\_[2]$, $b = -w\_[0]/w\_[2]$, $l = np.linspace(-1, 1)$.

And here is the final plot with test points (PLA plane defined in green). You can see that the PLA misclassified one test point (blue point over the line).

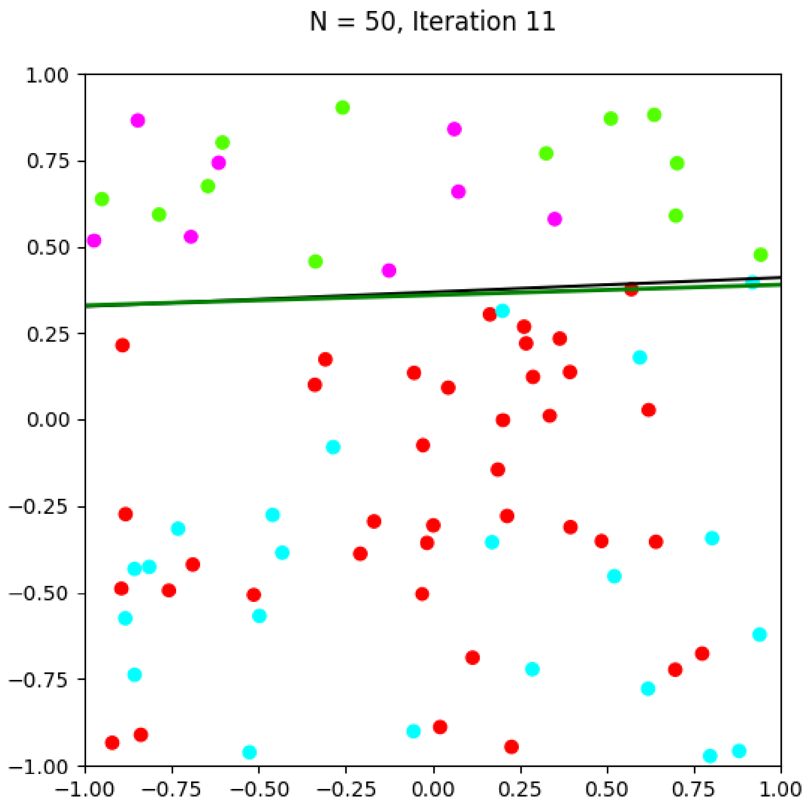


Figure 2: Training points shown in green / red. Testing points shown in purple / blue.

Variations

8) Initial Choice of Weights

Step size, $c = 0.01$.

Zeros

Zeros	
Iterations	**Misclassified Test Pts**
37	0
14	1
91	0
9	0
46	0

Our "control." Weights initialized at zeros gives the quickest, most accurate results.

Ones

Ones	
Iterations	**Misclassified Test Pts**
258	1
225	0
224	0
230	2
376	0

Weights initialized at ones takes much longer, with similar accuracy.

random.uniform(-1,1)

random.uniform(-1,1)	
Iterations	**Misclassified Test Pts**
45	1
185	0
122	1
94	4
203	3

Randomly chosen weights run somewhere in between, with much lower accuracy.

9) Initial Choice of Step Size Constant (c)

Weights initialized at zeros.

Step size: 0.01

c=0.01	
Iterations	**Misclassified Test Pts**
15	0
31	0
27	2
30	0
80	1

Our "control." Quick results, fairly accurate.

Step size: 0.1

c=0.1	
Iterations	**Misclassified Test Pts**
27	0
165	0
335	0
13	1
16	0

May take much longer, however very accurate.

Step size: 1.0

c=1.0	
Iterations	**Misclassified Test Pts**
7	1
94	0
33	1
20	1
133	2

Usually quick, but not as accurate.

10) Order of Picking Points

Weights initialized at zeros. Step size, $c = 0.01$.

Random Misclassified Point

Random	
Iterations	**Misclassified Test Pts**
21	0
19	0
45	0
19	3
64	0

Our "control." Quick results, fairly accurate.

First Misclassified Point

First Point	
Iterations	**Misclassified Test Pts**
25	1
77	0
17	0
21	1
341	0

May take much longer due to an outlier, however very accurate.

Last Misclassified Point

Last Point	
Iterations	**Misclassified Test Pts**
47	0
45	1
20	2
45	1
60	1

Very steady quickness, but not as accurate.