Homework 1

Helen Ngo

September 20, 2016

Problem 1.2 Consider the perceptron in two dimensions: $h(\mathbf{x}) = \mathbf{sign}(\mathbf{w^T x})$ where $\mathbf{w} = [\mathbf{w_0}, \mathbf{w_1}, \mathbf{w_2}]^\mathbf{T}$ and $x = [1, x_1, x_2]^T$.

(a) Show that the regions on the plane where $h(\mathbf{x}) = +\mathbf{1}$ and $h(\mathbf{x}) = -\mathbf{1}$ are separated by a line.

(b) Draw a picture for the cases $\mathbf{w} = [\mathbf{1}, \mathbf{2}, \mathbf{3}]^\mathbf{T}$ and $\mathbf{w} = -[\mathbf{1}, \mathbf{2}, \mathbf{3}]^\mathbf{T}$.

Solution:

(a) Since $h(\mathbf{x}) = \mathbf{sign}(\mathbf{w^T x})$ determines the classification of a point, there exits a line

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

that is in neither region and separates the two classes. Rearranging the equation of the line into the x_2 intercept form [1], $x_2 = mx_1 + b$, we have

$$\begin{aligned} w_2 x_2 &= -w_1 x_1 - w_0 \\ x_2 &= -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}, \end{aligned}$$

where the slope $m = -\frac{w_1}{w_2}$ and intercept $b = -\frac{w_0}{w_2}$.

(b) Let $\mathbf{w} = [\mathbf{1}, \mathbf{2}, \mathbf{3}]^\mathbf{T}$. Then $h(\mathbf{x}) = \mathbf{sign}(\mathbf{w^T x})$

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{sign}([1, 2, 3]^T \mathbf{x}) \\ h(\mathbf{x}) &= \mathbf{sign}(x_0 + 2x_1 + 3x_2). \end{aligned}$$

Therefore the slope and intercept of the line are $-\frac{2}{3}$ and $-\frac{1}{3}$ respectively.

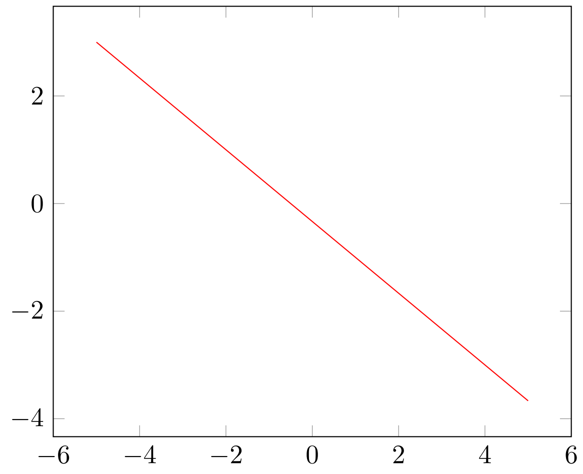
[1] Alternate form as states in the textbook: $x_2 = ax_1 + b$.

Similarly, let $\mathbf{w} = -[\mathbf{1},\mathbf{2},\mathbf{3}]^{\mathbf{T}}$. Then $h(\mathbf{x}) = \mathbf{sign}(\mathbf{w^Tx})$

$$h(\mathbf{x}) = \mathbf{sign}(-[1,2,3]^T\mathbf{x})$$

$$h(\mathbf{x}) = \mathbf{sign}(-x_0 - 2x_1 - 3x_2).$$

The slope and intercept of the line are also $-\frac{2}{3}$ and $-\frac{1}{3}$ respectively. Thus both cases would have the separating line:



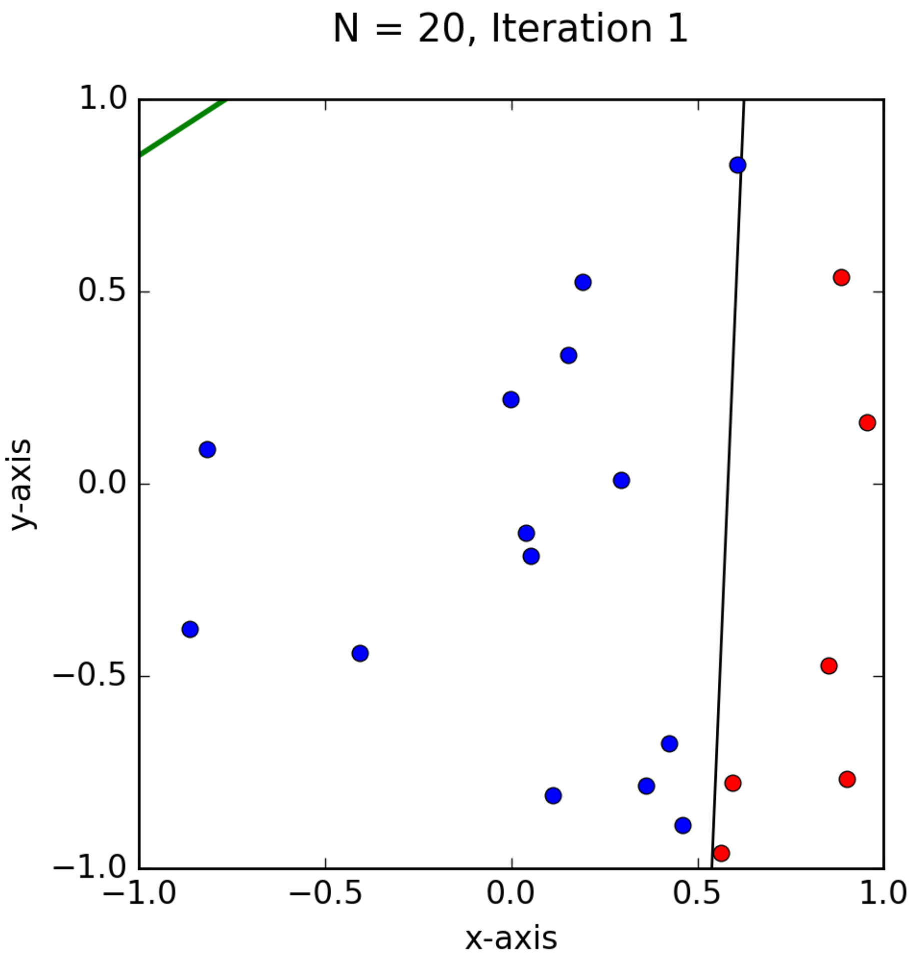
Problem 1.4 In Exercise 1.4, we use an artificial data set to study the perceptron learning algorithm. This problem lead you to explore the algorithm further with data sets of different sizes and dimensions.

Solution:

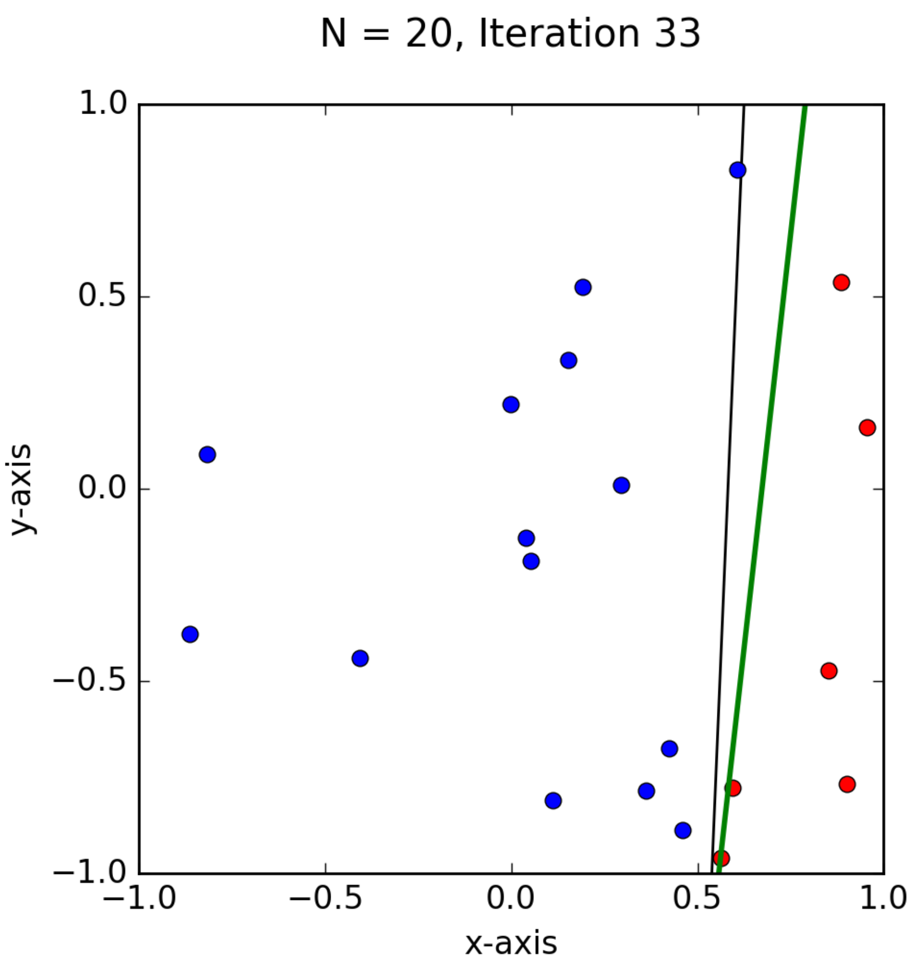
(a) Calling the perceptron learning algorithm code [1] with

```
def main():
    p = Perceptron(20)
    p.plot()
    p.pla(save=True)


main()
```

The first iteration produces a graph with a data set of size 20 with a black target function f separating the red and blue classes.

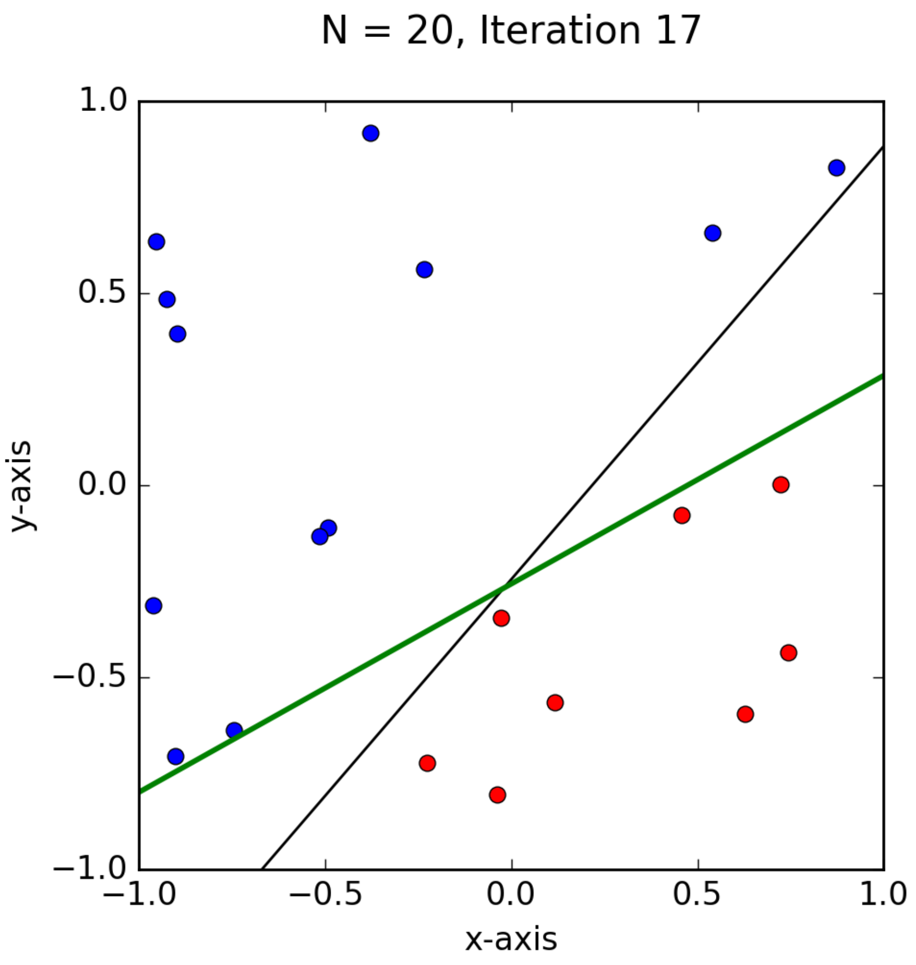


(b) Using the algorithm from the previous part, 33 iterations were completed before finding the final hypothesis g for the same set of 20 points. The final hypothesis function is shown in green.



Although the final hypothesis function g is relatively close to the target function f, it should be noted that the points from the two regions are close to each other, and that all final hypothesis functions with no misclassifications would be close to the target function.

(c) Running the perceptron learning algorithm again, with another randomly generated data set of size 20, the final hypothesis function g_1 and target function f_1 against the data set are shown below.



It can be seen that g_1 is not as close to f_1 as g is to f. Since the sets of data is different, it makes sense that their final hypothesis functions would have a different relationship to their respective target functions.

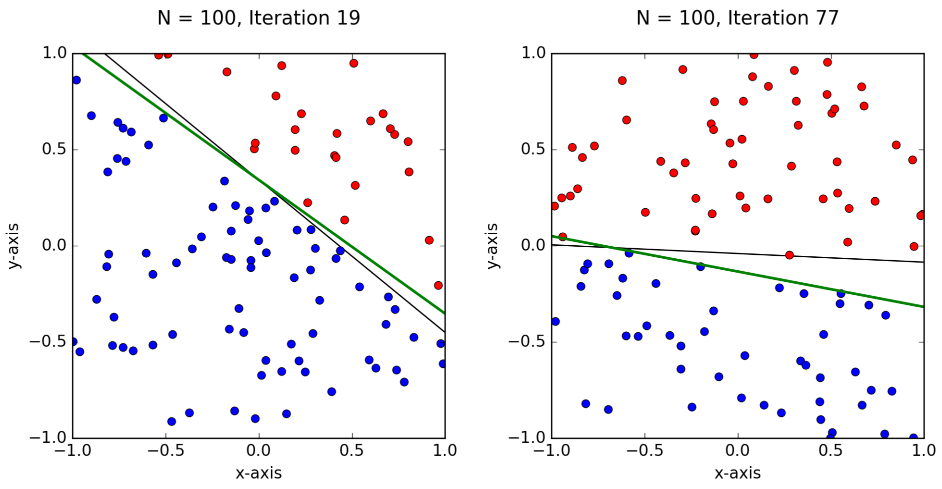
(d) Before running the algorithm for a randomly generated data set of size 100, the "pla" function in algorithm was modified so that only the last iteration would be saved and the iteration count printed.

```
def pla(self, save=False):
    # Initialize the weigths to zeros
    w = np.zeros(3)
    X, N = self.X, len(self.X)
    it = 0
    # Iterate until all points are correctly classified
    while self.classification_error(w) != 0:
        it += 1
        # Pick random misclassified point
        x, s = self.choose_miscl_point(w)
        # Update weights
        w += s*x
    # Plot and save the last iteration.
    if save:
            self.plot(vec=w)
            plt.title('N = %s, Iteration %s\n' \
                        % (str(N),str(it)))
            plt.xlabel('x-axis')
            plt.ylabel('y-axis')
            plt.savefig('p_N%s_it%s' % (str(N),str(it)), \
                        dpi=200, bbox_inches='tight')
    # Print iteration count.
    print(it)
    self.w = w
```

The main function was also modified so that the perceptron generates a data set of size 100.

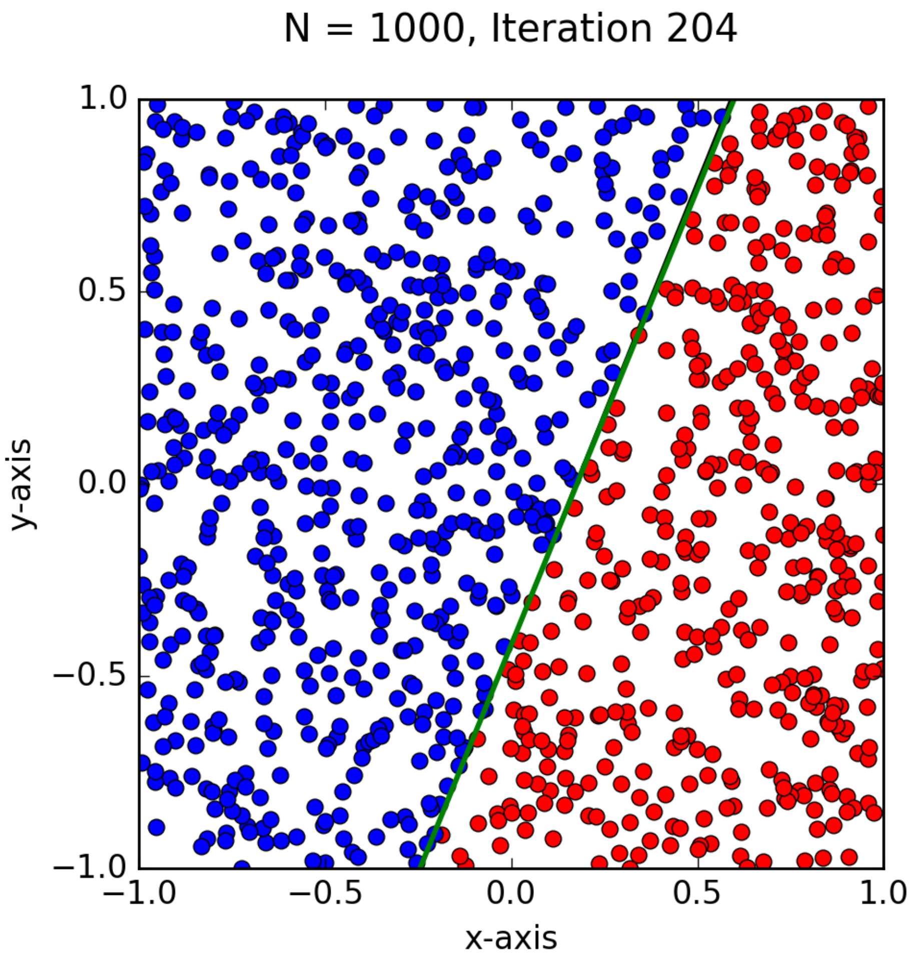
```
def main():
    p = Perceptron(100)
    p.plot()
    p.pla(save=True)


main()
```

The experiment was repeated twice since the first trial surprisingly concluded after only 19 iterations. The second trial concluded after 77 iterations.



Due to the wide range of results and the small sample size of trials, it is hard to compare the results for $N = 20$ to $N = 100$. Based solely on these four trials, the range of iterations until the final hypothesis function is found is greater for $N = 100$ than $N = 20$, and it generally takes more iterations for $N = 100$ to reach the final hypothesis.

(e) Repeating the perceptron for 1000 randomly generated points, it took 204 iterations for the perceptron to reach its final hypothesis, which is significantly more iterations than it took the perceptron to produce the final hypothesis g for $N = 20$. Also, g is closer to the target function than for previous sizes of data sets.

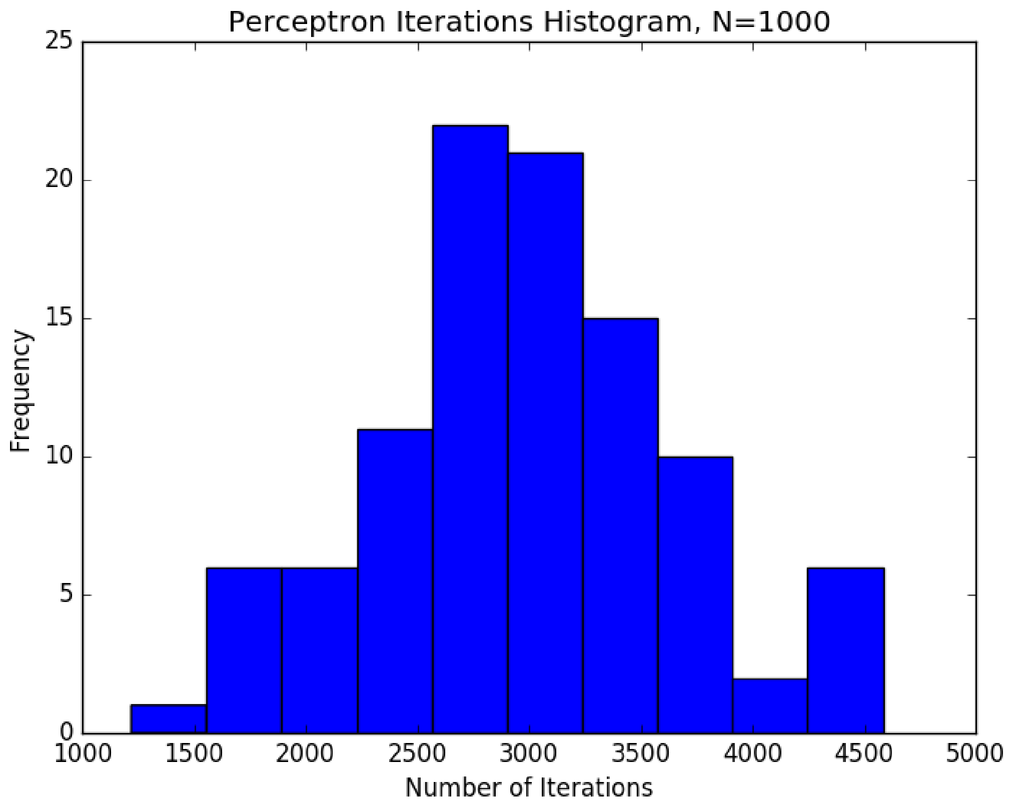


(f) The algorithm was modified so that it takes $x_n \in \mathbb{R}^{10}$ instead of $\mathbb{R}^2$ by altering the init and generate_points functions so that the target function is produced in ten dimensions. Using a randomly generated set of 1000 points in $\mathbb{R}^{10}$, it took 2600 iterations. The python script attached shows that modifications made to the mentioned functions.

(g) In order to repeat the algorithm on the same data as (f), a few modifications were made to the perceptron learning algorithm;the full script is attached. The main function was altered as shown below so that only one set of 1000 points are generated for 100 trials of the perceptron on those points. After all trials are completed, a histogram is produced for the number iterations.

```
def main():
    iterations = np.empty([100,1])
    p = Perceptron(1000)
    for i in range(100):
        iterations[i] = p.pla()
        print (iterations[i])
        print (i)
    plt.hist(iterations)
    plt.title("Perceptron Iterations Histogram, N=1000")
    plt.xlabel("Number of Iterations")
    plt.ylabel("Frequency")
    plt.savefig('Perceptron Iterations Histogram, N=1000')
    plt.show()

main()
```

A histogram of the iterations needed for a valid final hypothesis for the data set, such that each misclassified point is picked randomly is shown below.



(h) As the number of points N and dimensions d increases the the running time increases. As the number of dimensions increases, the accuracy decreases. Although no trials did not converge in the experiment conducted in (g), while experimenting with different data set of size 1000 in $\mathbb{R}^{10}$, the algorithm did not converge after 10000 iterations, which is significantly more iterations than trials that did converge under ten thousand iterations.

References

[1] https://datasciencelab.wordpress.com/tag/perceptron/