## 1: AML Pg. 34, Problem 1.4

**Answer:**

a) Figure 1 below shows the generated examples ($\mathbf{x}_n$, $y_n$), where $\mathbf{x}_n = (x_1, x_2)$, as well as the target function $f$, which is colored blue. Each example is labeled purple or yellow – purple if classified as $y_n = -1$, yellow if classified as $y_n = 1$.

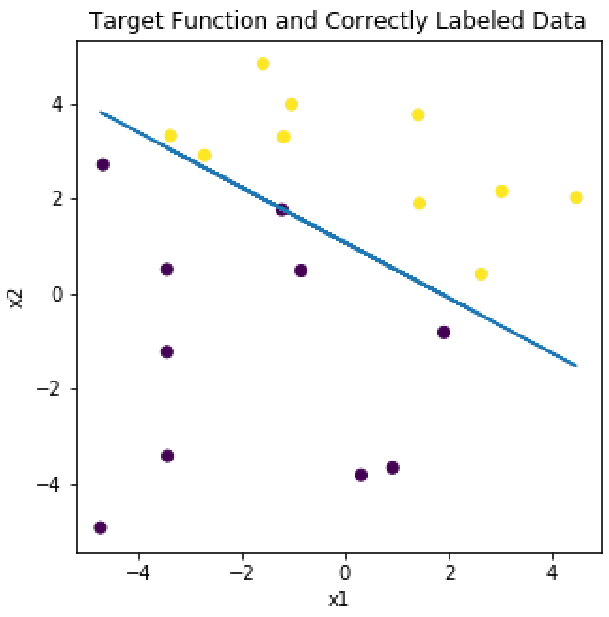

Figure 1: Plot of the target function and 20 data points with their classifications

b) When the perceptron learning algorithm (PLA) is run on the data set generated for a, it takes 19 updates to converge (linearly separating all data). Figure 1 is reproduced in Figure 2, but now with PLA's learned hypothesis $g$ given in orange. The hypothesis and target function are similar in y-intercept and slope. Interestingly, each component of the target function is roughly a factor of two different from that of the hypothesis; $g$: [-5.78276047 4.21619374 6.15622858], $f$: [-3.97303591 2.15931355 3.72110449]. This is acceptable, because the more negative bias counters the more positive other components of the hypothesis, producing similar classifications. Therefore, while the functions may be close in that they produce similar classifications, their weights may be very different.

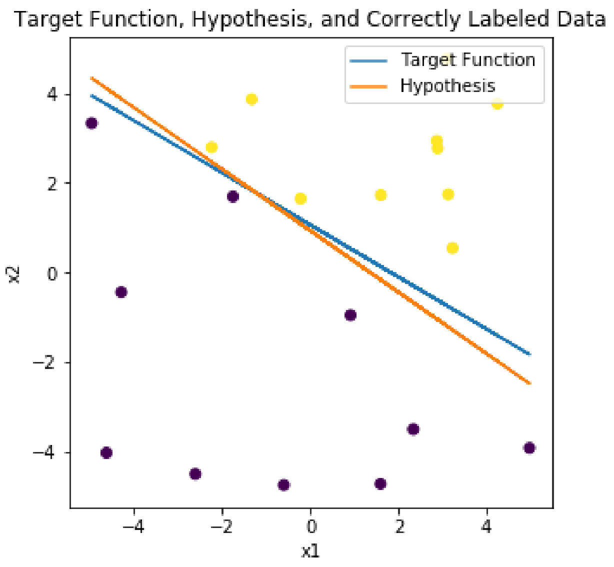

Figure 2: Plot of the same target function and data as part a, but now with a PLA-produced hypothesis $g$

c) Part b is repeated, but with a new dataset of 20 points separable by the same target function. Figure 3 gives the results. PLA takes 13 updates to converge – less than in part b. This is related to the difference in the the spreads of data in part b and part c. Looking at Figures 2 and 3, different classes are clearly more easily separable in the latter. This is not to say that more hypotheses can separate part c data than can separate part b data – floating-point weights ensure that the number of hypotheses is infinite in either case. Rather, randomly initialized hypotheses are more likely to separate part c data because of the spread between points of different classes. It follows that a lower number of updates is needed, on average, to separate part c data. This explains why the difference between target function and hypothesis is greater in part c than it is in part b; simply put, the hypothesis can be more different and still converge.

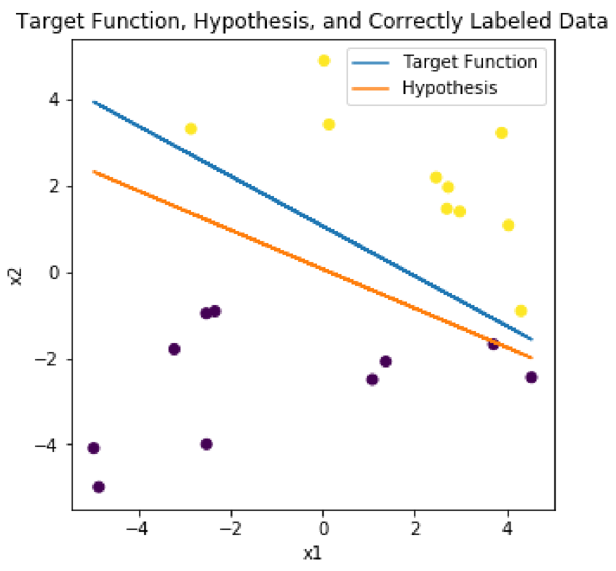

Figure 3: Plot of the same target function as part a, but now with new data and a PLA-produced hypothesis $g$

d) Part b is repeated, but with a new dataset of 100 examples separable by the same target function. Figure 4 gives the results. PLA takes 24 updates to converge. Following the part c discussion, this higher number occurs because of the relative closeness of differently classed points. This relative closeness simply comes from a higher number of data points generated. Each point generated has a certain probability of being a certain distance from a point of a different class. For small distances, that probability may be low, but a large number of points generated ensures that many points are close to points of the opposite class. As discussed, this increases the number of updates required, and makes the target function and hypothesis closer upon convergence.

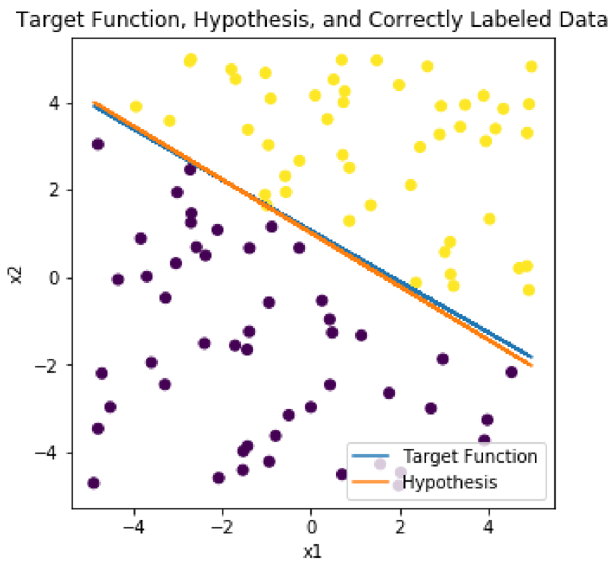

Figure 4: Plot of the same target function as part a, but now with 100 new examples and a PLA-produced hypothesis $g$

e) Part b is repeated, but with a new dataset of 1000 examples separable by the same target function. Figure 5 gives the results. PLA takes 213 updates to converge. The reason for this higher number of updates is the same as discussed in part d; namely, a higher number of points means more are likely to be close to ones of the opposite class. This reduces the probability of a randomly initialized hypothesis being correct, or indeed, that of one a few updates into PLA. That is, because data of opposite classes are closer, the target and hypothesis are closer upon convergence.

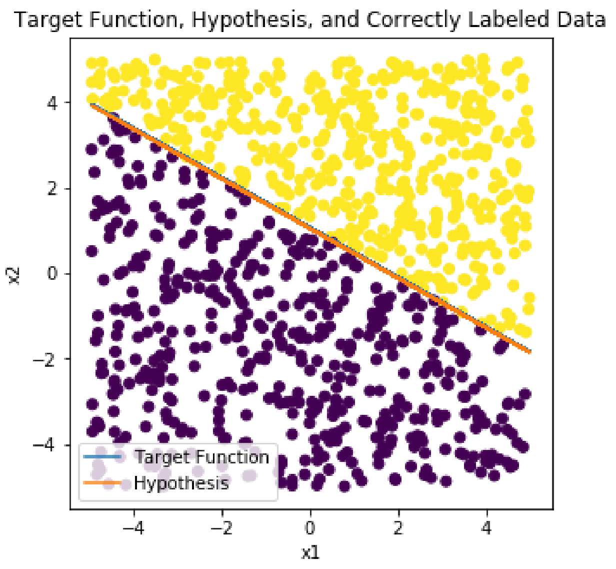

Figure 5: Plot of the same target function as part a, but now with 1000 new examples and a PLA-produced hypothesis $g$

f) I now generate 1000 new data points (same target function), each with a 10-dimensional input; that is, $\mathbf{x}_n = (x_1, x_2, ..., x_{10})$. PLA takes 2067 updates to converge. Despite having the same number of data points as part e, the part f PLA took much longer to converge. This arises from the higher dimensionality of part f inputs. To understand why, imagine two hyperplanes: one in a 3-D input space, and one in a 2-D input space. Given an equal number of input points, a slight perturbation (an update) to the 3-D hyperplane is likely to misclassify more points than an update would for the 2-D hyperplane. In the former, there are more degrees of freedom for the hyperplane to move in and therefore more points it could misclassify by moving. A weight update to a hypothesis for higher-dimensional input therefore causes more misclassifications than a weight update does for a two-dimensional input. This explains why the 10-dimensional input causes PLA to perform more updates than it does for 2-dimensional input – each update is less effective at achieving a net increase in number of points correctly classified.

g) I now repeat part f 100 times. The following histogram gives the number of experiments that correspond to each of 20 buckets. Each bucket denotes a number of updates. The histogram shows that the majority of experiments require between 1500 and 2500 updates to converge. The mean is 1950 updates. This affirms that the high number of updates required in part f was not abnormal.

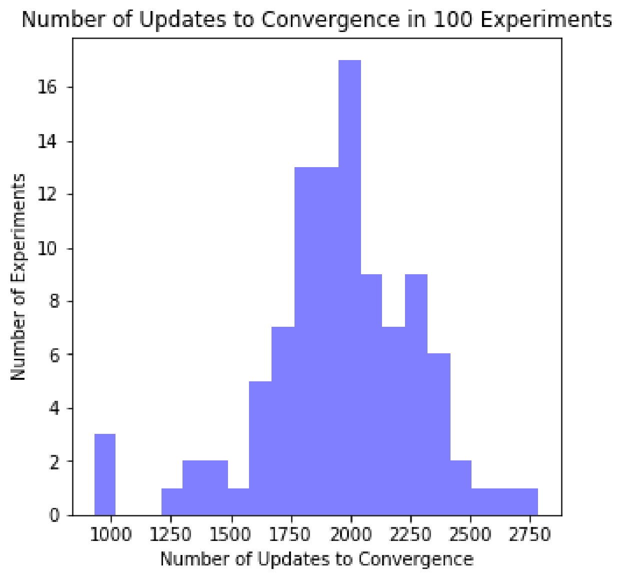

Figure 6: Histogram of common numbers of updates taken in PLA (part f repeated 100 times)

h) Since the data generated in this problem is all linearly separable, training accuracy is 100% regardless of $N$ and $d$. As for test set accuracy (not introduced in this problem), the answer is ”it depends”. If test data are drawn from the same distribution as training data, and the target function has no noise, the test data will be linearly separable in exactly the same manner, so test accuracy will be 100%. If, however, the target function has noise, some misclassifications may occur. As N increases, this will be a lower proportion of the total number of test data points, so test accuracy will increase. As d increases, more degrees of freedom allow noise to generate more non-linearly-classifiable points, possibly decreasing test accuracy.

Higher $N$ and higher $d$ make more updates necessary, increasing the runtime of PLA. This is explained in parts b-g. Likewise, making a prediction takes more time with higher $d$ because a dot product of two longer vectors implies more element-to-element multiplication and following summations.

## 2: AML Pg. 35, Problem 1.5

**Answer:**

a) NOTE: With $\eta = 100$, despite lowering initial weights and generated data significantly, I was unable to surpass a maximum of 140 updates without weights getting so large in magnitude as to overflow. I use a cutoff of 100 updates for part a.

Figure 7 shows 100 data points with their classifications: purple for -1, yellow for +1. The figure also shows a blue target function $f$ used to generate the data, and a modified-PLA-produced hypothesis $g$ in orange. I train this modified PLA with a learning rate $\eta = 100$ and a maximum of 100 weight updates. I then evaluate the learned hypothesis on test data. Test data consists of 10,000 examples generated via the same process as the training data. Testing error is 56.08%.

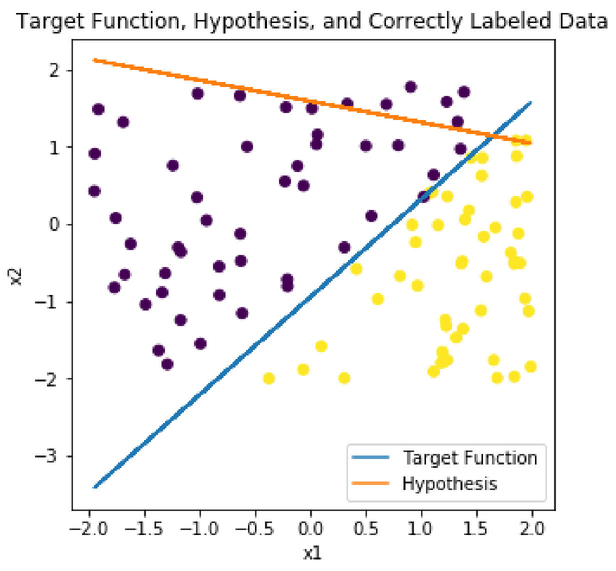

Figure 7: Plot of 100 correctly classified examples, the corresponding target function, and a modified-PLA-produced hypothesis $g$

b) I now reduce the learning rate to $\eta = 1$. Smaller weight adjustments allow me to increase the maximum number of updates to 1000. Testing error is 1.98%.

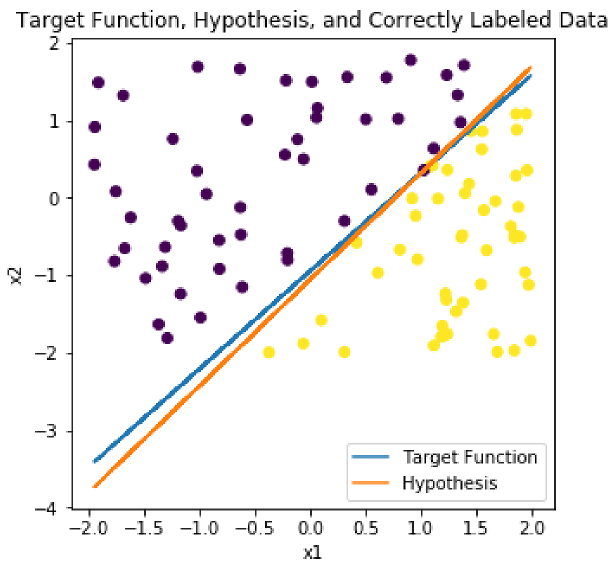

Figure 8: Same as plot in part a, but $g$ is now learned with $\eta = 1$ and 1000 updates as the maximum.

c) Again with a maximum number of updates of 1000, I reduce the learning rate further to $\eta = 0.01$. Testing error is 0.93%.

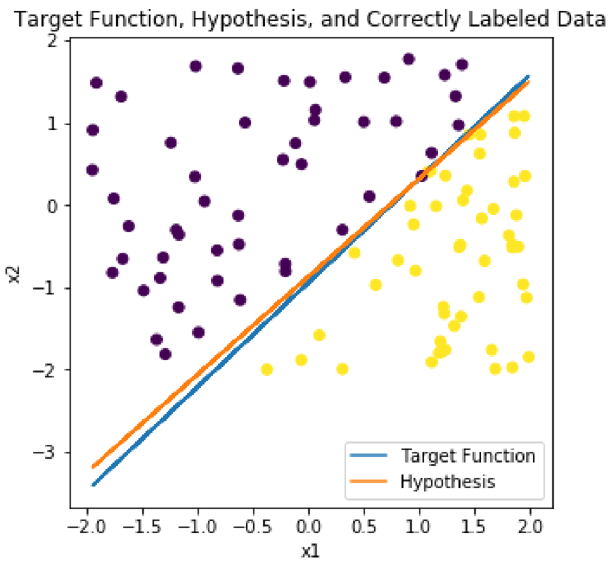

Figure 9: Same as plot in part a, but $g$ is now learned with $\eta = 0.01$ and 1000 updates as the maximum.

d) Again with a maximum number of updates of 1000, I reduce the learning rate further to $\eta = 0.0001$. Testing error is 28.39%.

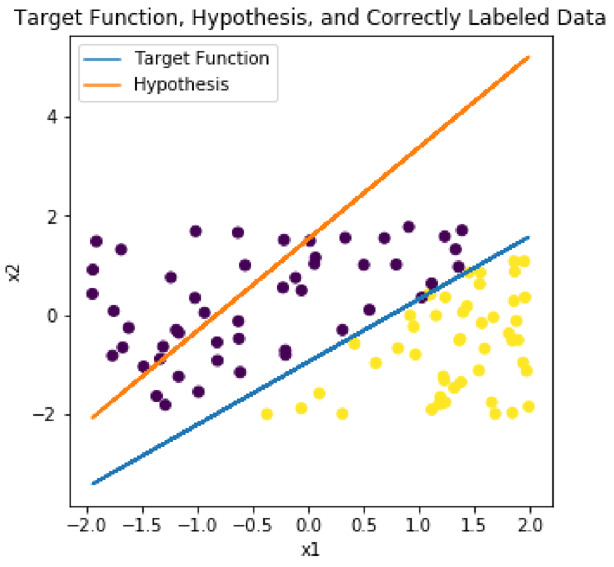

Figure 10: Same as plot in part a, but $g$ is now learned with $\eta = 0.0001$ and 1000 updates as the maximum.

e) From parts b to c, learning rate drops from $\eta = 1$ to $\eta = 0.01$, and test error decreases from 1.98% to 0.93%. From parts c to d, learning rate drops from $\eta = 0.01$ to $\eta = 0.0001$ and test error increases from 0.93% to 29.39%. In other words, for this data and a cutoff of 1000 training updates, test performance is not a monotonic function of learning rate: $\eta = 0.01$ performs best.

By contrast, with $\eta = 100$ and a cutoff of 100 training updates, test error is greater than that of random classification: 56.8%. One cannot fully say whether the higher learning rate causes this, as I had to decrease the cutoff by a factor of 10 in order to prevent large weights from causing overflow. However, repeating part b with a cutoff of 100 (second-to-last box in attached code), I achieve a test error of 2.53%. With equal cutoffs, the $\eta = 1$ modified PLA outperforms the $\eta = 100$ modified PLA substantially.

This suggests that learning rates much higher than $\eta = 1$ worsen performance drastically. For much higher learning rates, weight updates are too large – each update may cause more misclassifications than correct classifications. A too-high learning rate modifies the weights in each update so much as to never achieve convergence. Essentially, too-high learning rates cause weight updates that overshoot minimum training error.

As observed in the learning rate decrease from part c to part d, too-low learning rates can also worsen performance given a finite number of updates. With a small learning rate, each update does not modify the hypothesis much – possibly such that even the misclassified point motivating the update is not correctly classified after the update. Given an infinite number of updates and linearly separable data, however, a small learning rate will still converge. This is in marked contrast with the continually worsening behavior of a too-high learning rate. As Figure 10 shows, 1000 updates was not enough for the part d $\eta = 0.0001$ to converge on training data. However, when I increase the cutoff number of updates to 100,000, an $\eta = 0.0001$ modified PLA does converge and achieve a test accuracy of 1.04%. This is shown in Figure 11 (and thelast box in attached code).

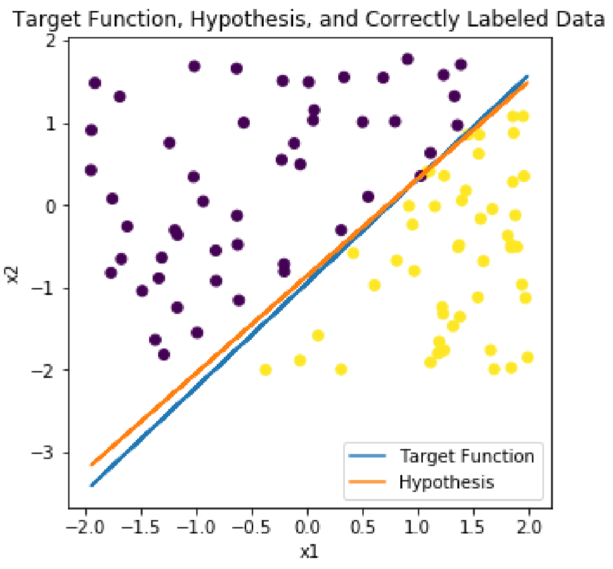

Figure 11: Same as plot in part a, but $g$ is now learned with $\eta = 0.0001$ and 100,000 updates as the maximum.