**Jason Harris**

**527334236**

# EEE 511
# Artificial Neural Networks
# Homework Number One

additional material is available at

http://www.geekspiff.com/academics/eee511/hw1

## Problem One

An odd sigmoid function is defined by

$$\varphi(v) = \frac{1-\exp(-av)}{1+\exp(-av)}$$

$$= \tanh\left(\frac{av}{2}\right)$$

To show that the derivative of $\varphi(v)$ with respect to $v$ is given by

$$\frac{d\varphi}{dv} = \frac{a}{2}\left[1-\varphi^2(v)\right],$$

I'll use several identities of hyperbolic trig function:

$$\tanh(u) = \frac{\sinh(u)}{\cosh(u)}$$

$$\frac{d\left[\sinh(u)\right]}{du} = \cosh(u) \qquad \frac{d\left[\cosh(u)\right]}{du} = \sinh(u)$$

Thus, taking the derivative gives

$$\frac{d\varphi}{dv} = \frac{\frac{a}{2}\cosh\left(\frac{av}{2}\right)}{\cosh\left(\frac{av}{2}\right)} - \frac{\frac{a}{2}\sinh^2\left(\frac{av}{2}\right)}{\cosh^2\left(\frac{av}{2}\right)}$$

$$= \frac{a}{2}\left[1-\tanh^2\left(\frac{av}{2}\right)\right]$$

$$= \frac{a}{2}\left(1-\varphi^2\right)$$

At the origin, the value of the derivative is

$$\frac{a}{2}\left[1-\left(\frac{1-1}{1+1}\right)^2\right] = \frac{a}{2}\ .$$

If the slope parameter *a* is made infinitely large, the slope becomes infinite at the origin and the sigmoid becomes

$$\varphi(v)\Big|_{a\to\infty} = 2u(v) - 1$$

where $u(v)$ is the Heaviside (threshold) function.

## Problem Two

Let the cost function of a (simple) network be $\mathsf{E}(\vec{w}) = w_1^2 + 30w_2^2 + 5w_1w_2$, where $\omega_1$ and $\omega_2$ are the components of a two-dimensional weight vector $\vec{w}$. The global minimum of this quadratic function occurs at $\vec{w}^* = [0,0]^{\mathrm{T}}$.

### *Part (a)*

I performed computations using the MATLAB script at
http://www.geekspiff.com/academics/eee511/hw1.
I began with a learning rate of $\eta = 0.05$. However, this was unstable and oscillated out of control (see Figure 1). I then investigated with the smaller learning parameters shown in Figures 2-4. There are larger versions of these plots on the web page.

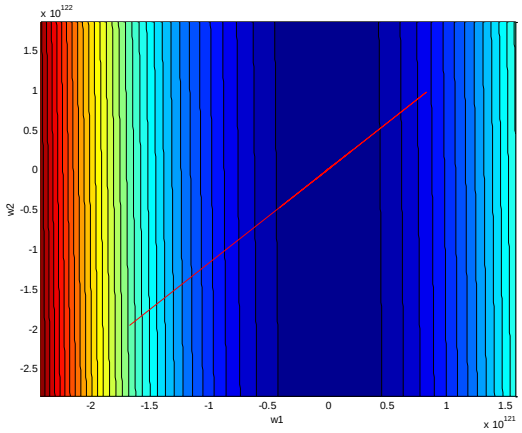


Figure 1: This simulation oscillated out of control, never converging. $\eta = 0.05$

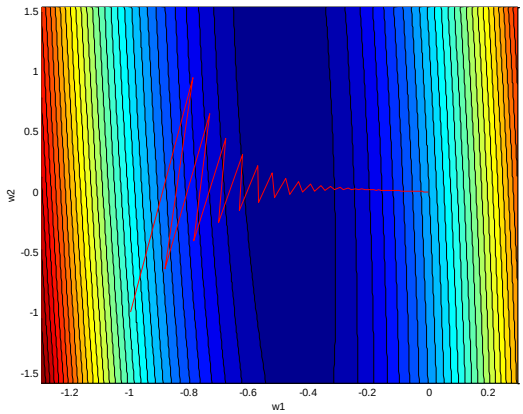


Figure 2: This simulation has a learning rate that causes it to oscillate to the minimum. $\eta = 0.03$, 127 iterations.

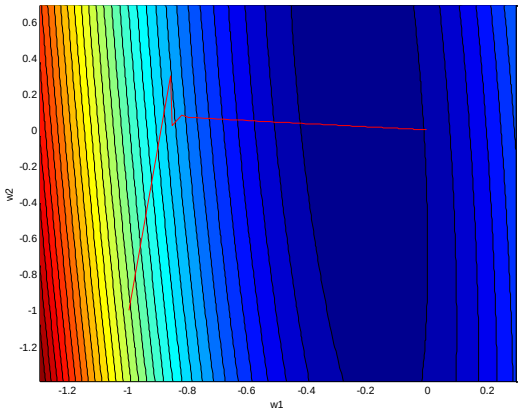


Figure 3: The learning rate is small enough in this simulation that instead of oscillating all the way to the minimum, it overshoots along one axis, corrects quickly, and then proceeds slowly along the remaining axis. $\eta = 0.02$, 179 iterations.

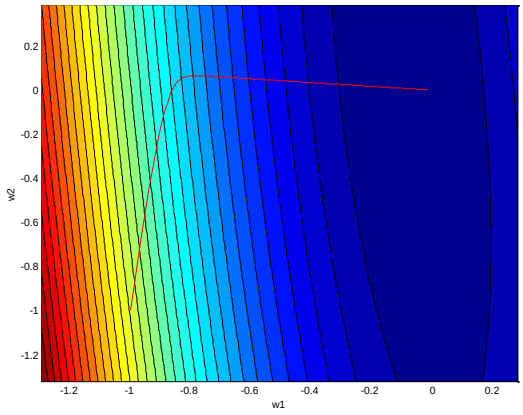


Figure 4: This simulation did not oscillate at all. $\eta = 0.005$, 543 iterations.

I ended the iteration after the change in the weight vector's magnitude became smaller than an arbitrary parameter. In this case, I used 0.001. However, for computations that didn't converge, I simply counted the number of iterations, and when the number exceeded an arbitrary constant, I halted execution.

## *Part (b)*

In this section, the same computation is performed, but 'momentum' is added to the steepest-descent algorithm. The code used in this section can be obtained on the web page. Figure 5 illustrates the change that the addition of 'momentum' can make. The calculation was performed using a learning parameter of $\eta = 0.005$. The red line, which is identical to Figure 4, is for the case in which the momentum weighting factor is $\mu = 0$. This run took 543 iteration. The green line shows the same simulation with $\mu = 0.8$. This run took only 128 iterations. Clearly, the addition of momentum is a good thing…

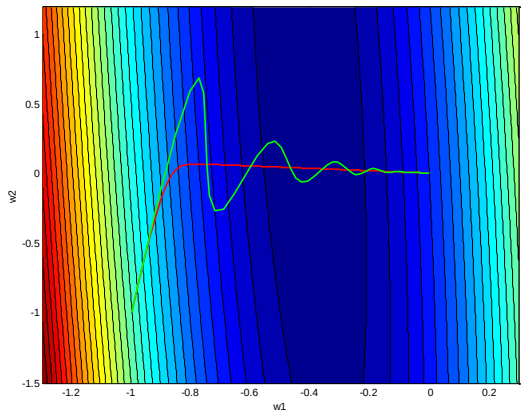


Figure 5: Steepest-descent with (green, 128 iterations) and without (red, 543 iterations) "momentum". $\eta = 0.005$.

### *Part (c)*

In this section, I computed the minimum of the cost function using Newton's method. In this method, the change in the weight vector is given by

$$\Delta\vec{w}(n) = -\mathbf{H}^{-1}(n)\,\vec{g}(n)$$

where $\mathbf{H}^{-1}(n)$ is the Hessian matrix of $\mathsf{E}(\vec{w})$.

Since the cost function is quadratic, a single application of Newton's method gives the global minimum. Hence, my code has no iteration functionality built in to it. The code is available on the web page.

## Problem Three

The error function is defined as

$$\mathsf{E}(\mathbf{w}) = \frac{\delta}{2}\left\|\mathbf{w} - \mathbf{w}(n)\right\|^2 + \mathsf{E}_0(\mathbf{w})$$

where the error function that was used in the pure Gauss-Newton approach is denoted by

$$\mathsf{E}_0(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)\ .$$

Our objective is to minimize the error. This accomplished by finding $\mathbf{w}^*$ such that

$$\nabla\mathsf{E}(\mathbf{w}^*) = 0\ .$$

Taking the derivative of the modified error function and setting it equal to zero gives

$$\nabla \mathsf{E}(\mathbf{w}) = \delta\left[\mathbf{w} - \mathbf{w}(n)\right] + \nabla \mathsf{E}_0(\mathbf{w})$$

$$\nabla \mathsf{E}(\mathbf{w}) = \delta\left[\mathbf{w} - \mathbf{w}(n)\right] + \mathbf{J}^{\mathbf{T}}(n)\mathbf{e}(n) + \mathbf{J}^{\mathbf{T}}(n)\mathbf{J}(n)\left[\mathbf{w} - \mathbf{w}(n)\right]$$

$$\left[\delta\mathbf{I} + \mathbf{J}^{\mathbf{T}}(n)\mathbf{J}(n)\right]\left[\mathbf{w} - \mathbf{w}(n)\right] + \mathbf{J}^{\mathbf{T}}(n)\mathbf{e}(n) = 0$$

Solving for $\mathbf{w}$ gives

$$\mathbf{w} = \mathbf{w}(n) - \left[\mathbf{J}^{\mathbf{T}}(n)\mathbf{J}(n) + \delta\mathbf{I}\right]^{-1}\mathbf{J}^{\mathbf{T}}(n)\mathbf{e}(n) \ .$$

This is the new value of $\mathbf{w}$ given in the Levenberg Marquardt scheme.

## Problem Four

The hyperapple has a radius *r* and the peel has a thickness $\varepsilon$.

### *Part (a)*

For the remainder of this problem, I'll use the term 'sphere' to denote an object that is spherically symmetric. Thus, a point is a zero-dimensional sphere, a line is a one-dimensional sphere, a circle is a two-dimension sphere, etc.

In addition, I'll be using the Euclidian distance. The Euclidian length of an *n*-dimensional vector $\vec{v}$ is $L = \sqrt{\langle \vec{v} | \vec{v} \rangle}$.

Generally, one can find the volume of an *n*-dimensional sphere by "slicing" it into thin (*n*-1)-dimensional strips and summing the volume of the strips needed to reconstitute the original shape. Figure 6 and Figure 7 illustrate this idea for two and three dimensional objects.

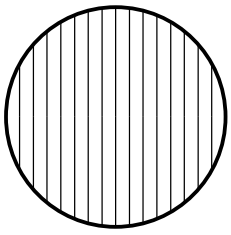

Figure 6: The "volume" of a two-dimensional object can be found by summing one-dimensional objects along the second dimension.

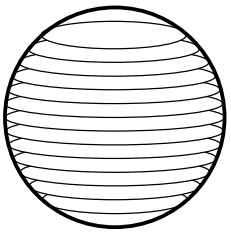

Figure 7: The volume of a three-dimensional object can be found by summing two-dimensional objects along the third dimension.

In order to accomplish this goal, we must integrate (*n*-1)-dimensional objects. Formulaically, this can be expressed as

$$V_n(r) = \int_{-r}^{r} dr'\, f_{n-1}\left(\sqrt{r^2 - r'^2}\right),$$

where $f_{n-1}(s)$ is the volume of the ($n$-1)-dimensional sphere obtained by slicing the $n$-dimensional object with a plane that is perpendicular to the $\hat{x}_n$ th axis and intersects this axis at a distance $s$ from the origin.

This quantity, $f_{n-1}(s)$, can be found by integrating the volumes of appropriately sized ($n$-2)-dimensional spheres. Figure 8 and Figure 9 show this principle for two- and three-dimensional spheres.

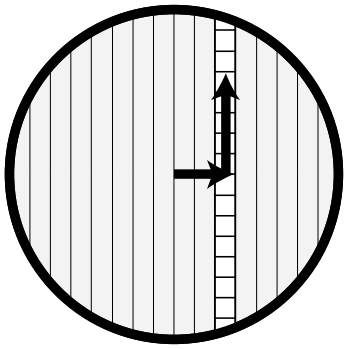

Figure 8: The volume of a two-dimensional sphere can be found by moving a distance $r$ along one axis and then summing the volumes of one-dimensional spheres along another axis.

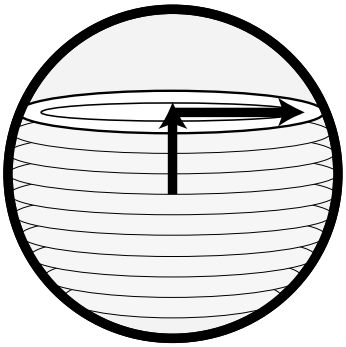

Figure 9: The volume of a three-dimensional sphere can be found by moving a distance $r$ along one axis and then summing the volumes of two-dimensional spheres along another axis.

Thus, the quantity $f_{n-1}(s)$ is (see Figure 10 and Figure 8)

$$f_{n-1}(s) = \int_{r'=-s}^{r'=s} ds' V_{n-2}(s') \ .$$

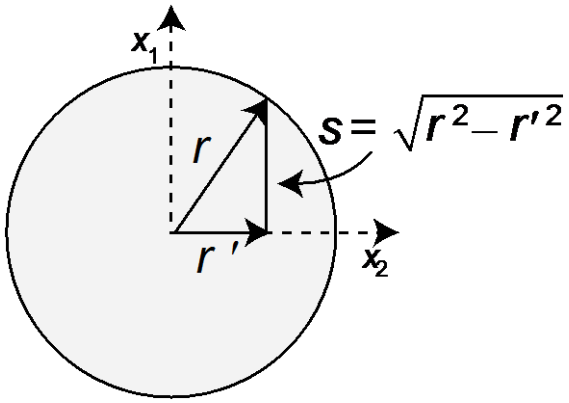


Figure 10: The radius of a one-dimensional sphere can be calculated by its distance from the origin.

Substituting this back in to the original expression gives

$$V_n(r)=\int_{-r}^{r}dr'\left(\int_{r'=-\sqrt{r^2-r'^2}}^{r'=\sqrt{r^2-r'^2}}ds'V_{n-2}(s')\right).$$

This can be expressed in polar coordinates as

$$V_n(r)=\int_0^r d\rho\int_0^{2\pi}\rho d\theta\ V_{n-2}(\rho),$$

where $\rho$ is the distance $r'$ along the $\hat{x}_n$ th axis and $\theta$ is the angle that $s'$ makes with the $\hat{x}_n$ th axis. Figure 11 illustrates this change.

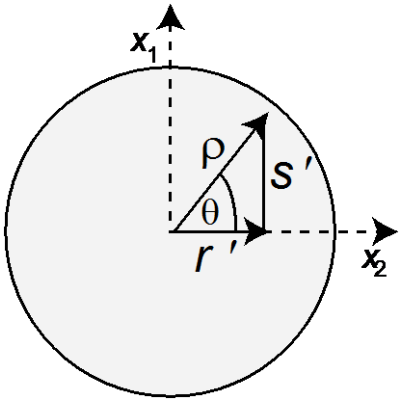


Figure 11: The two-dimensional sphere in polar coordinates.

Because the inner integral has no angular dependence, the final form of this expression is

$$V_n(r)=2\pi\int_0^r d\rho\ \ \rho V_{n-2}(\rho)\ .$$

We can verify this for the dimensions we know about. Beginning with $V_0(r)=1$ and $V_1(r)=2r$, we have

$$V_2(r)=2\pi\int_0^r d\rho\ \ \rho V_0(\rho)=2\pi\int_0^r rd\rho=\pi r^2$$

$$V_3(r)=2\pi\int_0^r d\rho\ \ \rho V_1(\rho)=4\pi\int_0^r \rho^2 d\rho=\tfrac{4}{3}\pi r^3\ .$$

In general, we have the relationship

$$V_n(r)=\begin{cases} \frac{1}{\pi r} & n=-1 \\ 1 & n=0 \\ (2\pi)^{\frac{n}{2}} r^n \prod_{i=1}^{\frac{n}{2}} \frac{1}{2i} & n \text{ even}, n>0 \\ \frac{1}{\pi}(2\pi)^{\frac{n+1}{2}} r^n \prod_{i=1}^{\frac{n+1}{2}} \frac{1}{2i-1} & n \text{ odd}, n>0 \\ \text{undefined} & \text{otherwise} \end{cases}.$$

It is interesting to note that we can find the volume of a (-1)-dimension sphere. In fact, we can find a volume for all odd, negative-dimensionality spheres. Unfortunately, we can't get a result for even, negative-dimensionality spheres because $V_{-2}(r)$ has the property

$$\frac{1}{2\pi}=\int_0^r d\rho\,\rho V_{-2}(\rho).$$

for which the only solution is $V_{-2}(r)=0$. But this solution would make all even, positive-dimensionality spheres have zero volume. Math that has no semblance to the real world is fun…

---

Moving on (and ignoring negative-dimensionality hyperspheres ☺), we can now easily express the fraction of the volume of the rest of the hyperapple to the volume of the peel (see Table 1). This expression is (for a *d*-dimensional hyperapple)

$$f_d\left(\frac{\varepsilon}{r}\right)=\frac{r^d}{(r+\varepsilon)^d-r^d}=\frac{1}{\left(1+\frac{\varepsilon}{r}\right)^d-1} \qquad d>0.$$

$$\varepsilon=0.02r$$

| **Dimensions (*d*)** | $f_d\left(\frac{\varepsilon}{r}\right)$ |
|---|---|
| 2 | 24.75 |
| 3 | 16.34 |
| 128 | 0.08611 |
| ∞ | 0 |

Table 1: The ratio of the apple's volume (without the rind), to the volume of the rind as the number of dimensions is increased.

Notice that since $1+\frac{\varepsilon}{r}>1$, when $d\to\infty$, the portion of the total volume contributed by the outer rind approaches the total volume.

---

If we have samples of data that are uniformly distributed throughout a hypercube, we know that the sample density is constant. In other words, given a subvolume $V$, the sample density $\frac{samples}{V}$ is constant, regardless of $V$. Using rectangular coordinates instead of polar coordinates, it is clear that the hypercube exhibits symmetry about the origin. Thus, the qualitative results obtained for the hypersphere also apply to the hypercube – namely, as the dimensionality increases, the portion of the total volume that is found in the outer portions approaches the total volume.

Thus, since the portion of the volume increases as distance from the origin is increased, the sample density must decrease. Most of the samples can therefore be found at the origin.

This was a cool problem, by the way.

# Problem Five

## *Part (a)*

The MATLAB script is fairly self-explanatory. See the script on the webpage for more details. The results obtained using the Least Squares Error method can be seen in Figure 12.

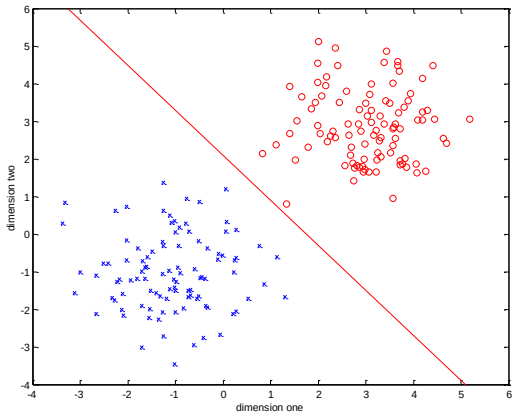


Figure 12: A discriminating line separating two linearly-classifiable sets of training data. This was generated using the Least-Squares Error method.

## *Part (b)*

This is slightly more tricky. The problem is figuring out when the computation has converged. The approach that I am taking is to present each training element once, in random order, and then determine whether the mean of the weight correction is smaller than a preset magnitude. If so, the simulation has converged.

I also tried using a running average. This, however, did not give good results. The accuracy and the number of iterations varied wildly depending upon the order in which the training samples were presented.

Simply checking the magnitude of the correction after each application of training data did not work, of course. If several similar training elements were presented consecutively, the computation finish successfully, when actually, the results were quite wrong.

Once I had settled upon the approach in which each element is presented once and then mean is inspected, I discovered that the LMS method is much more sensitive to the learning parameter than the Least Squares Error method. Table 2 illustrates this sensitivity.

| Learning Parameter | Convergence Magnitude | Mean Iterations |
|---|---|---|
| $1 \cdot 10^{-2}$ | 1 | 0 |
| $1 \cdot 10^{-2}$ | 0.01 | 22.2 |
| $1 \cdot 10^{-3}$ | 0.001 | 32.8 |
| $1 \cdot 10^{-4}$ | 0.0001 | 268.5 |
| $5 \cdot 10^{-4}$ | 0.0001 | 109.7 |

Table 2: Mean iteration count for various parameter settings.

Figures 13, 14 and 15 show the discriminating lines that were generated while finding the mean iteration count for each scenario. One can see that the scenarios that took more iterations resulted in greater accuracy. However, the simulation for which $\eta = 5 \cdot 10^{-4}$ is a good compromise between accuracy and speed.

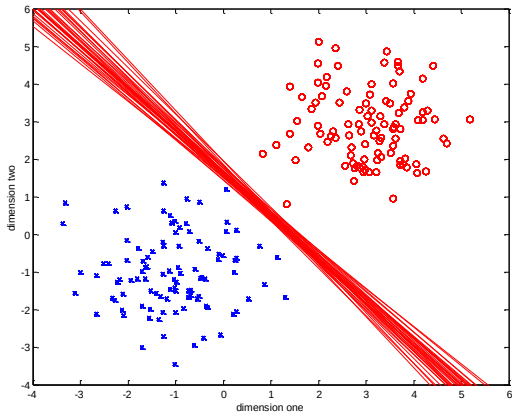


Figure 13: Discrimination lines when $\eta = 1 \cdot 10^{-2}$, convergence magnitude = 1.

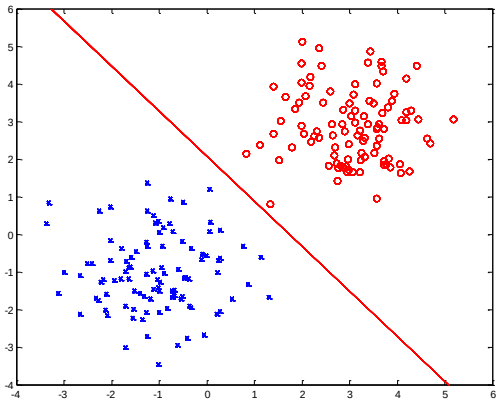

Figure 14: Discrimination lines when $\eta = 1 \cdot 10^{-4}$, convergence magnitude = 0.0001.

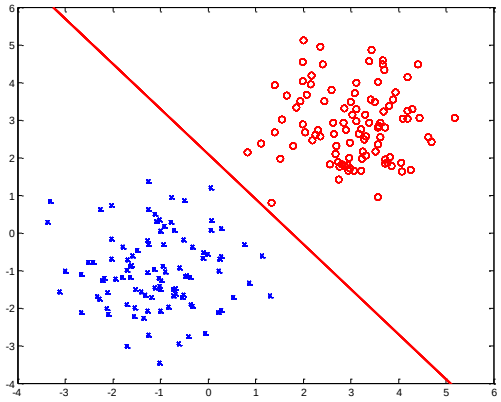

Figure 15: Discrimination lines when $\eta = 5 \cdot 10^{-4}$, convergence magnitude = 0.0001.

## *Part (c)*

I was unable to find a learning parameter value that did not correctly classify the training data within 2 or 3 iterations. I believe that the signum limiting function makes the Perceptron invariant to the scale of the learning parameter. Figure 16 shows a discriminating line generated using $\eta = 1 \cdot 10^{-20}$. The MATLAB script used to generate this result is available on the web page.

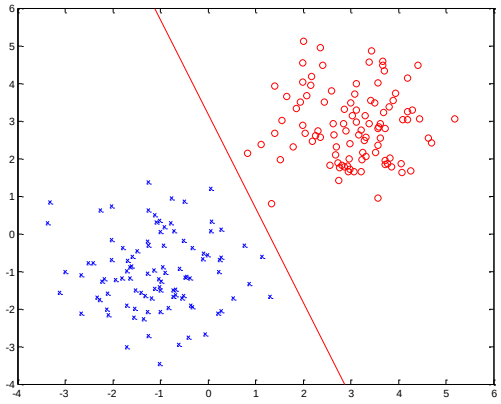

Figure 16: Classification using Perceptron learning.

## *Part (d)*

I used the functions `tic` and `toc` to measure the execution time of each of the three computation methods. Table 3 shows these execution times. Surprisingly, Perceptron learning produced the quickest results. However, I speculate that with a more difficult distribution of training data, things will change.

| Computation Method | Execution Time [s] |
|---|---|
| Least Squares Error | 0.016 |
| Least Mean Squares | 0.984 |
| Perceptron Learning | 0.015 |

Table 3: Execution time for various classification methods.

## *Part (e)*

Now the classes are not linearly-separable. The Least Squares Error algorithm does not iterate, so its solution time is not changed. It seems to place the discrimination line normal to the direction of highest sample density (see Figure 17).

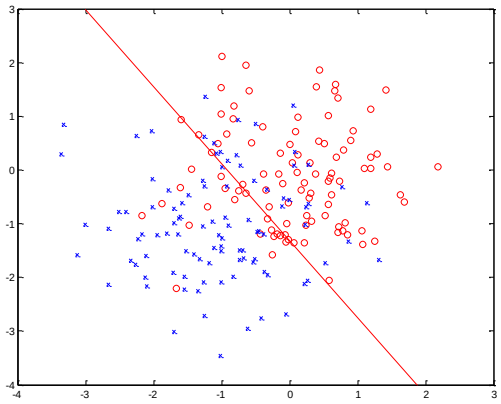


Figure 17: The discrimination line using Least Squares Error for classes that are not linearly-separable. The discrimination line seems to be orthogonal to the direction of highest sample density.

The LMS algorithm produced roughly the same discriminating line as the Least Squares Error method, but it took nearly double the time that it took for the linearly-separable case (1.796 s). Figure 18 shows the discrimination line for this case.

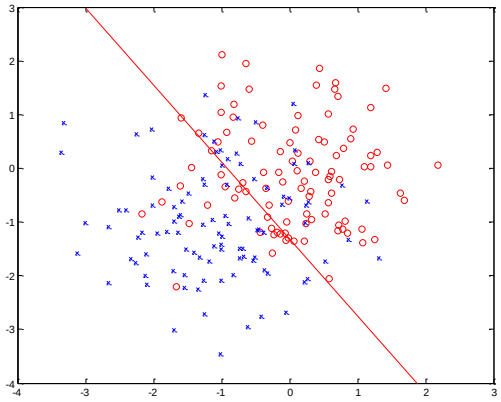


Figure 18: The discrimination line for linearly-non-separable classes using the LMS method.

The Perceptron cannot obtain a solution. Since it iterates until every element of the training set has been classed correctly, and the training set is not linearly-separable, the Perceptron can never converge.

To summarize, for this simple case, the Least Squares Error method is the clear performance winner. It consistently gives the best discrimination line in the quickest time. However, if the classes were not stationary, it is quite possible that the LMS method would become useful, as it can respond more quickly to changing conditions.