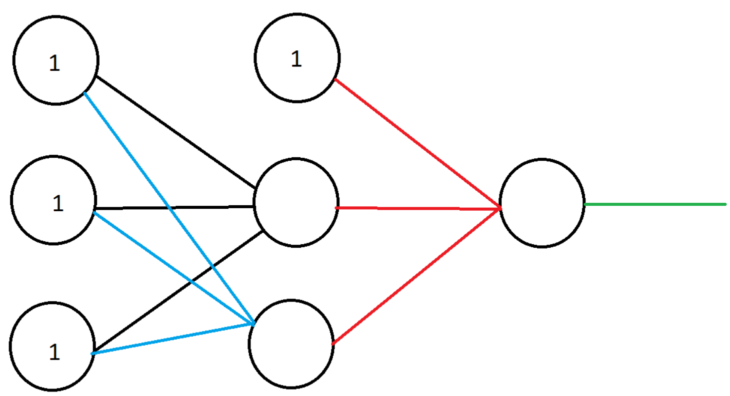
# Assignment 12

Darwin Ding

December 7, 2016

## 1. Neural Networks and Backpropagation




(a)

This is a really crude drawing of the neural net for this part. For the sigmoid transformation tanh(x), we end up with $x_0 = [1, 1]$, $s_1 = [tanh(.75), tanh(.75)] = [.635148, .635148]$, $x_1 = [1, .635148, .635148]$, $s_2 = [.56757]$, $h(x) = x_2 = \mathbf{.5135757}$.

Initializing for back-propagation for tanh, we get $\delta_2 = 2(.5135757 - 1)(1 - .5135757^2) = \mathbf{-.71625}$. $\delta_1 = \mathbf{[-0.10683, -0.10683]}$

Now we can easily attain the partial derivatives for the final step: $\frac{de}{dW_1}$, $\frac{de}{dW_2}$ and thus the final gradient (which is just the set of the partial derivatives) are below. Note that "d1" and "d2" correspond to $\delta_1$ and $\delta_2$ respectively.

```
x2: 0.513575735687
d2: -0.716249965414
d1: [-0.10682614113705637, -0.10682614113705637]
de/W1:
[[-0.10682614 -0.10682614]
 [-0.10682614 -0.10682614]
 [-0.10682614 -0.10682614]]
de/W2:
[[-0.71624997]
 [-0.45492542]
 [-0.45492542]]
```

Instead of displaying the entirety of the math for the identity as opposed to the tanh function, below are all of the relevant values. Note that, again, the gradient is simply the set of the partial derivatives below.

```
C:\Users\Darwin\Document
x2: 0.625
d2: -0.75
d1: [-0.1875, -0.1875]
de/W1:
[[-0.1875 -0.1875]
 [-0.1875 -0.1875]
 [-0.1875 -0.1875]]
de/W2:
[[-0.75  ]
 [-0.5625]
 [-0.5625]]
```

(b) Below are the values for the tanh transform neural net with slightly modified weights of .2501. There is pretty negligible difference. Again, d1/d2 mean $\delta_1$ and $\delta_2$.

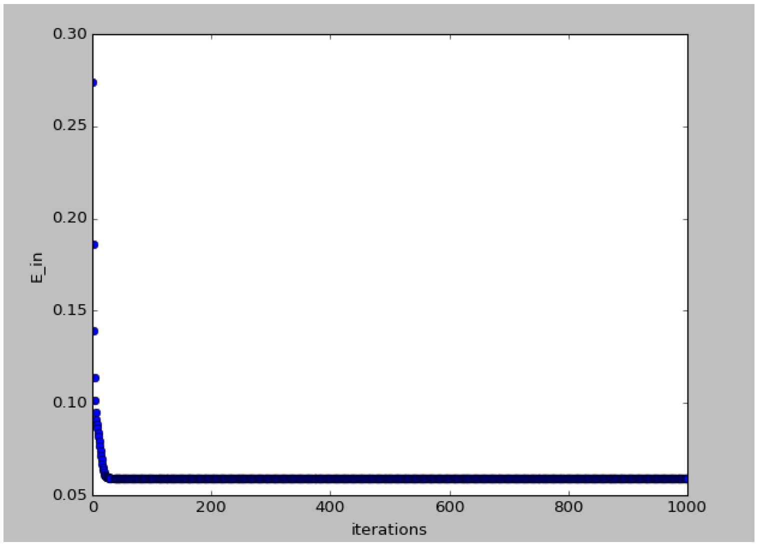
```
x2: 0.513808744547
d2: -0.7156740862
d1: [-0.10674225507214183, -0.10674225507214183]
de/W1:
[[-0.10674226 -0.10674226]
 [-0.10674226 -0.10674226]
 [-0.10674226 -0.10674226]]
de/W2:
[[-0.71567409]
 [-0.45468771]
 [-0.45468771]]
```

Additionally, here are the same weights with the identity transform neural net.

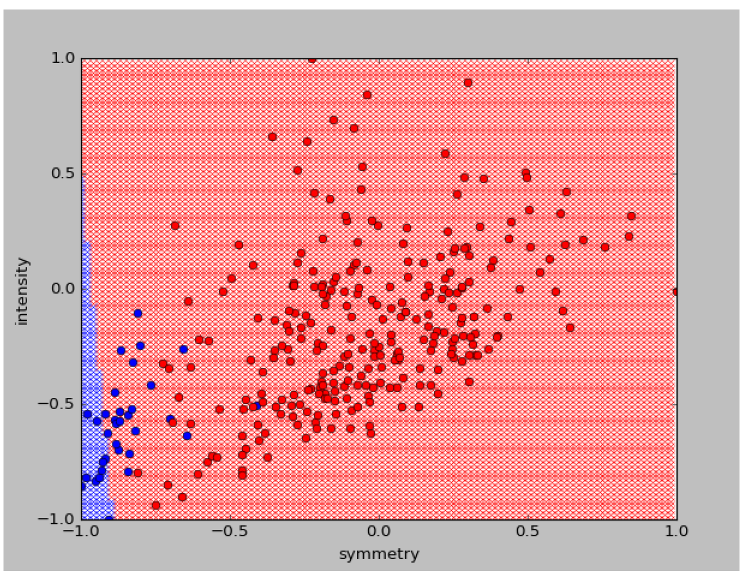
```
x2: 0.62540006
d2: -0.74919988
d1: [-0.18737488998799998, -0.18737488998799998]
de/W1:
[[-0.18737489 -0.18737489]
 [-0.18737489 -0.18737489]
 [-0.18737489 -0.18737489]]
de/W2:
[[-0.74919988]
 [-0.56212467]
 [-0.56212467]]
```

# 2. Neural Network with Digits

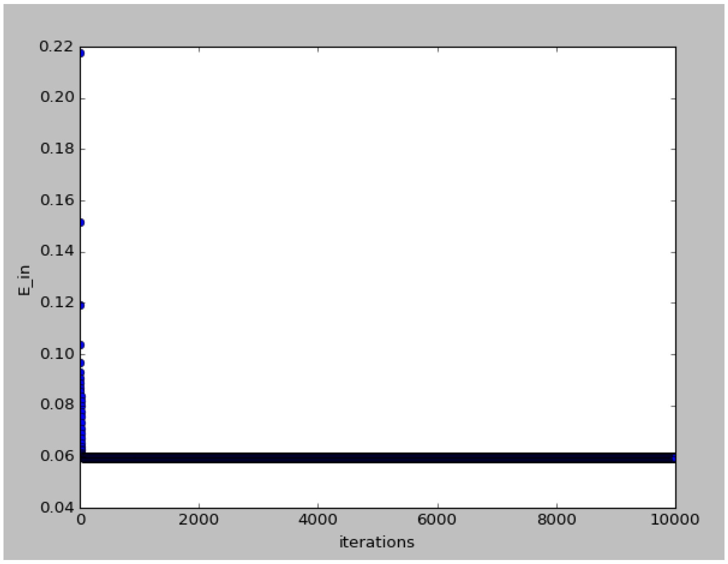
(a) Below is the graph of $E_{in}$ versus iterations. The variable gradient descent was run up to 2 million iterations, but I typically found it converging around 10000.



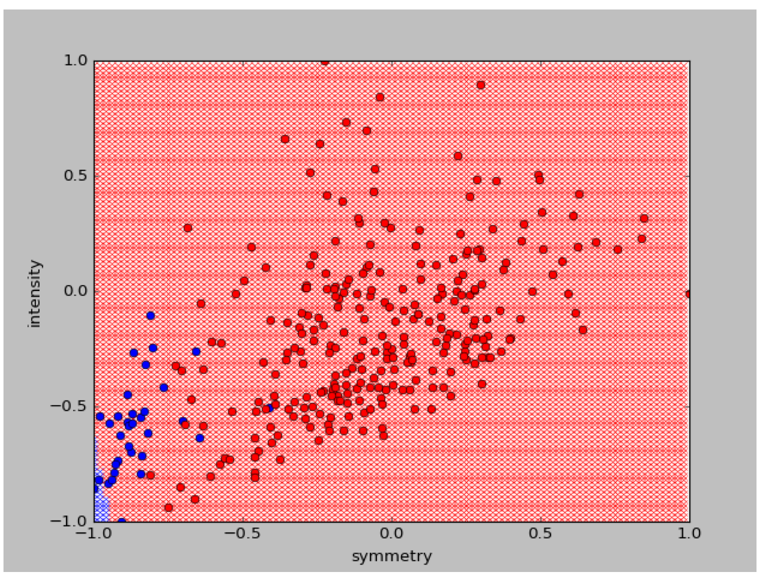

Additionally, below is the decision boundary. It is doing some classification in a strangely linear fashion, but isn't too shabby. It still has an $E_{in}$ of **0.0586**.




(b) Below is the graph of $E_{in}$ versus iterations for the variable gradient with a weight decay as specified in the problem description. It's hard to see on the graph but this graph converges a little earlier than the other one.

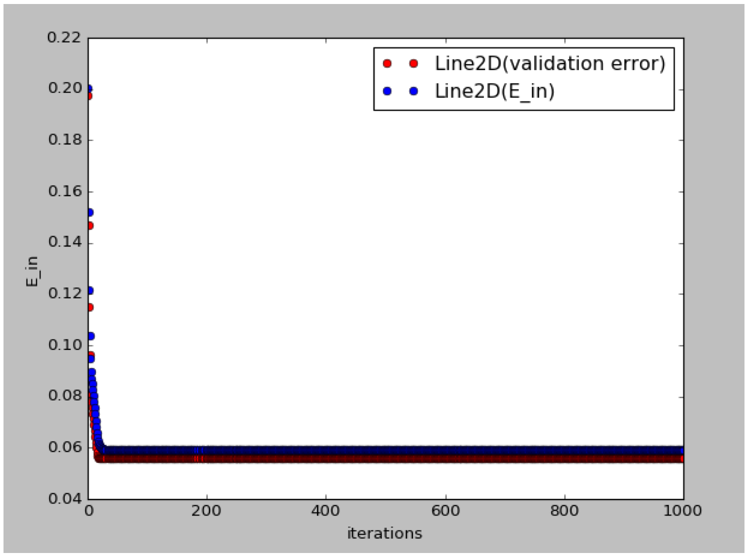



Additionally, below is the decision boundary. The $E_{in}$ is a little worse now, at **0.05947**. The classifier has become a little worse, but at least we've made some gain from regularizing.

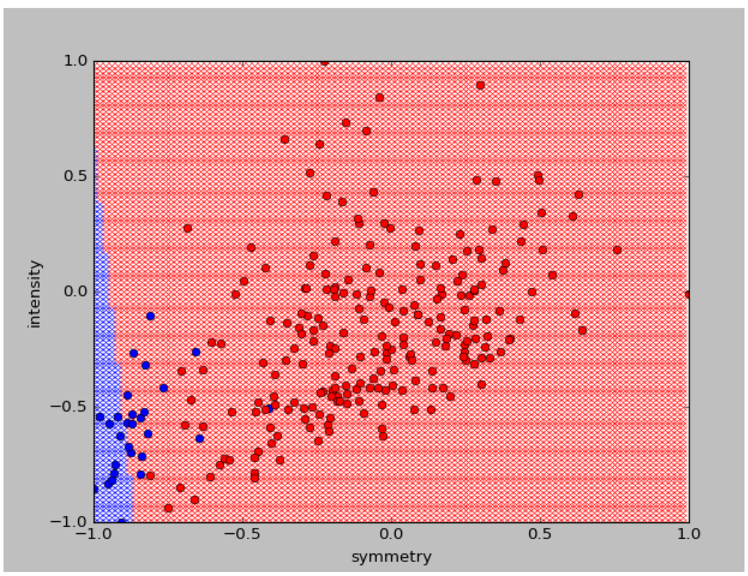



(c) I ran the algorithm with an implemented validation and training set, and found that the validation set error was converging just like the training set. However, it did go up very marginally after a certain point, and had a local minimum at $E_v = \mathbf{.055}$.

Below is a graph of the validation set error (red) and the $E_{in}$ error (blue) over time:




And then additionally, the final graph using the weights with lowest validation error instead of the actual error, which got me an $E_{in}$ of **.0592**.



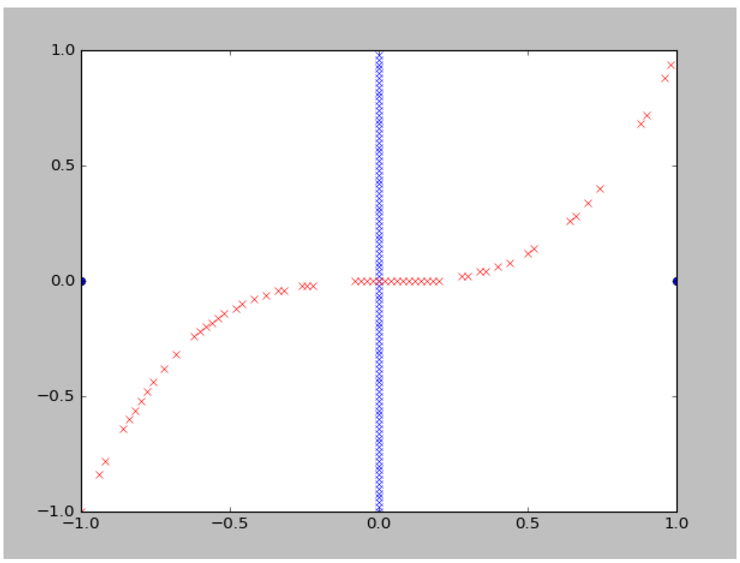

The error is a very marginally worse using this algorithm because it relies on the validation set error instead of the $E_{in}$ error, but we can trust that it has better generalization error because of the randomly assorted test set.

## 3. Support Vector Machines

(a) A hyper-plane in this 2-dimensional space is simply a line, and it is pretty intuitive that the best way to split the two points with the largest cushion is by drawing the line $\boldsymbol{x} = \mathbf{0}$. By moving the line any closer to either point, you cushion the other point harder but lower the overall cushioning by more than you gain.

(b) In this new space, [1, 0] becomes [1, 0] and [-1, 0] becomes [-1, 0]. The points are overall unchanged mostly due to the nature of 1s and 0s with exponents. Again, the optimal hyperplane here will be $\boldsymbol{z_1} = \mathbf{0}$, but this will not end up being a line in X-space.

(c) Below are the classifiers in X and Z space. The blue line is the $\boldsymbol{x} = \mathbf{0}$ line, and the red line is the $\boldsymbol{z_1} = \mathbf{0}$ one in Z-space. The two blue points on either side are the actual points in X and Z-space.




(d) After transforming $[x_0,\ x_1]$ and $[y_0,\ y_1]$, we get $[x_0^3 - x_1,\ x_0 x_1]$ and $[y_0^3 - y_1,\ y_0 y_1]$. The dot product of these two is:

$$[(x_0^3 - x_1)(y_0^3 - y_1), x_0x_1y_0y_1]$$
$$= x_0^3y_0^3 - x_0^3y_1 - x_1y_0^3 + x_1y_1 + x_0x_1y_0y_1$$

(e) The kernel function as a classifier when written in functional form can really be boiled down to $h(x) = sign(x^3 - y)$ for any point (x, y).