CS 4804 Homework 6
Solution Sketches

---

1. (15 points) For n input boolean variables, the number of boolean functions possible is 2^{2^n} (given in your book on page 761). To see why, notice that a truth table for n variables has 2^n rows. Each of these rows can have a 1 or a 0 as the value of the boolean function. Thus, the total number of functions is 2^{2^n}. For 2 variables, the answer is 16 and for 3 variables, the answer is 256.

   In the case of 2 variables all but two are linearly separable and can be learned by a perceptron (these are XOR and XNOR). In the case of 3 variables, all but 152 of them are linearly separable. Alex Hanisch's solution elegantly shows why (it is self-explanatory):

```
#include <iostream.h>

void main()
{
//                 w-------x
//                / |     /|
//              a-------b  |
//              |   |   |  |
//              |   y---|--z
//              | /     | /
//              c-------d
//
        int a,b,c,d,w,x,y,z, count=0;
        bool bad;
        for(int i=0; i<256; i++)
        {
                z=i%2;
                y=i%4/2;
                x=i%8/4;
                w=i%16/8;
                d=i%32/16;
                c=i%64/32;
                b=i%128/64;
                a=i/128;

                bad=false;

                if(a==d && b==c && a!=b)
                        bad=true;
                if(a==y && w==c && a!=w)
                        bad=true;
                if(w==z && x==y && w!=x)
                        bad=true;
                if(x==d && b==z && x!=b)
                        bad=true;
```

```
                if(a==x && b==w && a!=w)
                        bad=true;
                if(y==d && z==c && d!=z)
                        bad=true;

                if(a==z && c==x && a!=x)
                        bad=true;
                if(b==y && d==w && b!=w)
                        bad=true;
                if(a==z && b==y && a!=y)
                        bad=true;
                if(c==x && d==w && c!=w)
                        bad=true;
                if(a==z && w==d && a!=w)
                        bad=true;
                if(c==x && b==y && c!=y)
                        bad=true;

                if(bad)
                        count++;
//                      cout<<a<<b<<c<<d<<w<<x<<y<<z<<endl;
        }
        cout<<count<<endl;
}
```

This program prints 152 as output. So, there are 14 linearly separable functions of 2 variables and 104 linearly separable functions of 3 variables.

2. (15 points) Notice that the training examples are *all the same* but sometimes the neural network is expected to produce 1 and sometimes the correct answer is 0. There is no way the network will learn this dataset correctly! This form of contradictory and noisy data should lead the neural network algorithm to find some intermediate solution. We are also not given the network topology (but the question is only interested in finding the output, not the weights).

   The error function can be written as (t_i is the target, expected output for the ith example and o_i is the actual output):

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=1}^{100}(t_i - o_i)^2 \\ &= \frac{1}{2}\left(\sum_{i=1}^{80}(1 - o_i)^2 + \sum_{i=1}^{20}(0 - o_i)^2\right) \\ &= \frac{1}{2}\left(\sum_{i=1}^{80}(1 - o)^2 + \sum_{i=1}^{20}(0 - o)^2\right) \end{aligned}$$

   where the last simplification is because the output will be the same for all cases (since the input is the same).

Differentiating E w.r.t. o we get:

$$\frac{\partial E}{\partial o} = 100\,o - 80$$

Setting the derivative to zero, we find that $o = 0.8$. In other words, the network is learning the observed frequency of the output values.

3. (20 points) If you train the perceptron on the given data, you will find that it learns the given classifications but doesn't quite converge to the voting function described in the problem. For instance, the weights will not correspond meaningfully to the relative importances of the votes of the inputs. This is because the voting function is a boolean function and there are 2^{2^6} boolean functions of six inputs. This number is greater than 10^{19}. There is no way we can figure out the right function from this huge space giving only 14 examples! You can determine if your perceptron has learned the 'right' function by testing it on new data (you can cook these up, since we have told you how the data should be generated). If given substantially more data, you will find that the perceptron will indeed learn the desired voting function.

4. (30 points) The error function for a single instance is given by:

$$\begin{aligned} E &= \frac{1}{2}(\text{target} - \text{output})^2 \\ &= \frac{1}{2}(\text{target} - (w_0 + w_1 x + w_2 x^2 + w_3 x^3))^2 \end{aligned}$$

Our job is to get a learning rule to update the weights. Differentiating E w.r.t. the weights, we get:

$$\begin{aligned} \frac{\partial E}{\partial w_0} &= -(\text{target} - \text{output}) \\ \frac{\partial E}{\partial w_1} &= -(\text{target} - \text{output})\,x \\ \frac{\partial E}{\partial w_1} &= -(\text{target} - \text{output})\,x^2 \\ \frac{\partial E}{\partial w_1} &= -(\text{target} - \text{output})\,x^3 \end{aligned}$$

This gives us a learning rule for the wacky perceptron as:

$$\begin{aligned} w_0 &= w_0 + \eta\,(\text{target} - \text{output}) \\ w_1 &= w_1 + \eta\,x\,(\text{target} - \text{output}) \\ w_2 &= w_2 + \eta\,x^2\,(\text{target} - \text{output}) \\ w_3 &= w_3 + \eta\,x^3\,(\text{target} - \text{output}) \end{aligned}$$

where we have added the η term to only 'slightly nudge' the weights and since we are moving in the direction of decreasing E, the weight update has a positive sign. Using this learning rule, you can learn the weights for the given data.

FYI, the data was generated by the following MATLAB code:

```
xrange = 0:0.01:1; a = [];
for i=xrange
        a = [a;0.3 + i + 3*i*i*i];
end
```

Your learned weights must thus correspond to $w_0 = 0.3, w_1 = 1, w_2 = 0, w_3 = 3$.

5. (20 points) Trivial. The exact weights will depend on your choice of initial weights, the order in which the examples were presented, the learning rate, and whether you used an input encoding of (0 and 1) or (1 and -1).