CPSC 633-600 Spring 2013 (Total 100 points)
Machine Learning
Homework 1

See the course web page for the **due date** and **submission info**.

Instructor: Yoonsuck Choe

1 Supervised Learning

Problem 1 (Written: 10 pts):

Theorem: [Haussler, 1988].

> If the hypothesis space H is finite, and D is a sequence of $m \geq 1$ independent random examples of some target concept c, then for any $0 \leq \epsilon \leq 1$, the probability that the version space with respect to H and D is not ϵ-exhausted (with respect to c) is less than
>
> $$|H|e^{-\epsilon m}$$

This bounds the probability that any consistent learner will output a hypothesis h with $error(h) \geq \epsilon$

If we want this probability to be below δ

$$|H|e^{-\epsilon m} \leq \delta \tag{1}$$

then

$$m \geq \frac{1}{\epsilon}(\ln |H| + \ln(1/\delta)) \tag{2}$$

(1) Show how we can get to Eq. 2 from Eq. 1. Show all the steps.

(2) Discuss how m increases or decreases as ϵ and δ change.

	Mark one choice	Discuss the implications
If ϵ increases	then m will (↑, ↓)	
If δ increases	then m will (↑, ↓)	
Given a fixed m, if δ decreases	then ϵ can (↑, ↓)	

Problem 2 (Written: 5 pts):

When $VC(H)$ increases,

(1) Representational power of the hypothesis (increases, decreases).

(2) Sample complexity (increases, decreases).

(3) Explain why the two, representational power and sample complexity, pose a trade-off.

2 Perceptrons

Problem 3 (Written: 10 pts): Consider the data set below.

(1) Can a single perceptron learn the following training set without error?

(2) Can a two layer perceptrons learn the following training set without error? If so, what is the minimum number of hidden units needed to achieve this? If not, explain why.

x	y	class
0	0	-1
0	1	-1
0	2	-1
0	3	-1
0	4	-1
1	0	-1
1	1	-1
1	2	+1
1	3	-1
1	4	-1
2	0	-1
2	1	+1
2	2	+1
2	3	+1
2	4	-1
3	0	-1
3	1	+1
3	2	+1
3	3	+1
3	4	-1
4	0	-1
4	1	-1
4	2	-1
4	3	-1
4	4	-1

3 Gradient Descent

Problem 4 (Written: 5 pts): Given an error function

$$E(w) = (w-1)(w-2)(w-3)(w-7),$$

find:

$$\frac{\partial E(w)}{\partial w}.$$

Note that this is simply an ordinary derivative $\frac{dE}{dw}$.

Problem 5 (Written: 5 pts): How many minima does $E(w)$ have? You can draw a rough plot from $w = 0$ to 7.

Problem 6 (Written: 5 pts): With $\frac{\partial E(w)}{\partial w}$ calculated above, if you want to adjust w to minimize $E(w)$, what should Δw be? Write the answer as a polynomial function of w. (This is trivial, given the answer to problem 4.)

Problem 7 (Program: 15 pts): Using the gradient found above, write a short program to

1. Initialize w to a particular value w_0.
2. Repeat
$$w \leftarrow w + \eta \Delta w$$
until difference in successive error values $E(w)$ is less than 0.001. Set $\eta = 0.01$.

Experiments: Run your program with $w_0 \in \{0.5, 1.5, 2.3, 2.8, 4, 7\}$, and report the following:

1. Plot the function $E(w)$ in the background, and plot the changing $(w, E(w))$ positions as points in the same plot. See figure 1.
2. Report how many iterations were needed for $E(w)$ to reach the minima.

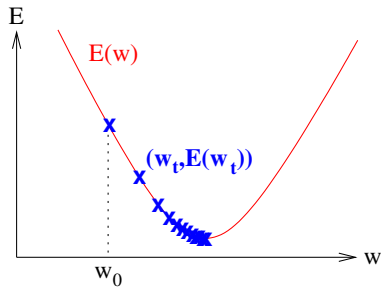


Figure 1: **Gradient Descent.** Note that $E(w)$ shown here in the figure is different from the one in problem 2. This is just an example.

Problem 8 (Program: 10 pts): Add momentum to the previous program and repeat the experiment, with $w_0 = 0.5$. Start with momentum rate of 0.1, and increase it by 0.1 until you can reach the global minumum. Report the momentum rate value that helped you get over the local mimimum.

Repeat the same experiment with $w_0 = 2.3$, and report your finding. The behavior will be different from the first experiment with $w_0 = 0.5$. Explain why you think this is the case.

Problem 9 (Written: 5 pts): Using chain rule, find the gradient for the function $\sigma(E(w))$ where $\sigma(x) = 1/(1+\exp(-x))$ and $E(w)$ is defined above:

$$\frac{d\sigma(E(w))}{dw}.$$

Write the answer as a function of w. **Hint:** $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$.

Problem 10 (Written: 10 pts): Consider these two functions of two variables:

$$E_1(w_1, w_2) = {w_1}^2 + {w_2}^2 \quad (3)$$

$$E_2(w_1, w_2) = {w_1}^2 + \frac{{w_2}^2}{2} \quad (4)$$

1. Illustrate by hand (or use Octave/Matlab) the contour plot of these two functions. Make the x and y range go from -10 to 10, and draw contour lines for $E_i(w_1, w_2), i = 1, 2$ values in the increment of 20.

2. Derive the gradient $\nabla E_1(w_1, w_2)$ and $\nabla E_2(w_1, w_2)$.

3. Pick the contour line at E value (z-axis value) of 40, and draw the gradient vector at 8 different points on that contour (evenly spaced around the contour), based on your derived $\nabla E_1(w_1, w_2)$ and $\nabla E_2(w_1, w_2)$.

4. Starting from $(5, 7)$, manually perform (you may use a calculator/program) the first four steps of gradient descent (try minimizing E_i while adjusting w_1 and w_2), with learning rate $\eta = 0.1$, so that

   $$\Delta\vec{w} = -\eta\nabla E_i(w_1, w_2), \text{for } i = 1, 2,$$

   where $\vec{w} = (w_1, w_2)$. Illustrate your results and show the calculated values of $\vec{w}$ over time.

5. Discuss how the behavior differs for the two different functions.

4 Backpropagation

Problem 11 (Program: 10 pts): Implement backprop (or use matlab neural network toolbox, etc.) and train it on the data set shown in Problem 3. Experiment with different network configurations:

(1) Single layer

(2) Two layer: 2 hidden units, 3 hidden units, ...

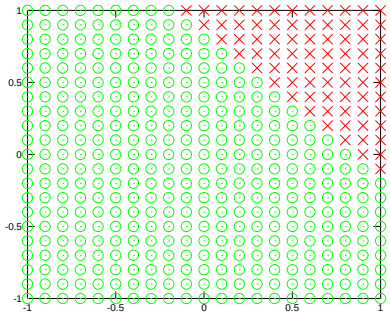
(3) Analyze the weights and plot the decision boundary (you'll get a bunch of line equations). Or, alternatively, you can plot the activation value of each hidden or output unit (if you had 3 hidden and 1 output unit, you will need to show 4 such plots in total). For the input, generate a mesh grid of finer grain than the grid in Problem 3 data set (see Octave/Matlab example below).

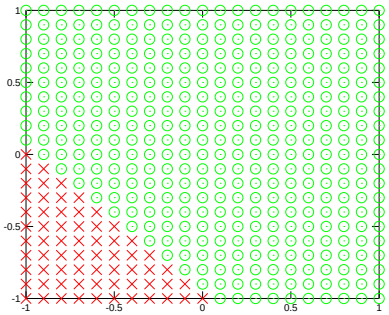
```
This shows how to create the input vectors in a
systematic manner. The example is shown in Octave.
"octave:1>" etc. is the command prompt.
```

```
octave:1> [x,y] =meshgrid([0:2])
x =

   0   1   2
   0   1   2
   0   1   2

y =

   0   0   0
   1   1   1
   2   2   2

octave:2> input_data = [vec(x),vec(y)]
input_data =

   0   0
   0   1
   0   2
   1   0
   1   1
   1   2
   2   0
   2   1
   2   2

# For actual poltting, use the following so that you have a finer grid.

octave:3> [x,y] =meshgrid([0:0.1:2])

# Now, [x,y] will go in as the input, and suppose you got it
# back in "output".
# Assume all vectors are column vectors.
# Also, the "activate( ... )" function is not well defined here.

octave:4> output = activate([x,y], weight_matrix, ...);

# If the output was in range [0..1], you can find the index for the
# outputs that exceed a certain threshold (here 0.5) and those that
# do not.

octave:5> pos_idx = find(output>=0.5);
octave:6> neg_idx = find(output<0.5);

# Now, you can plot the partitions.

octave:7> plot( xy(pos_idx,1),xy(pos_idx,2),"go",
```

```
 xy(neg_idx:1),xy(neg_idx,2),"rx" );

# "go" means "green" and "o" shaped markers.
# "rx" means "red" and "x" shaoed markers.
```

Examples of the above plot are shown below. These are from hidden unit activations that learned the XOR function.





Software downloads

You may use one of the following (first and second option recommended):

- C++ version: fully functional. `http://courses.cs.tamu.edu/choe/13spring/633/src/backprop-1.6.tar.gz`
- Octave version: key lines missing (by design). You need to fill in about 3 lines. `http://courses.cs.tamu.edu/choe/13spring/633/src/bp.m`
- Implement from scratch (whatever language).
- Matlab Neural Networks Toolbox.