HOMEWORK 3
PAC, VC DIMENSION, NEURAL NETWORKS

CMU 10-701: MACHINE LEARNING (FALL 2016)
https://piazza.com/class/is95mzbrvpn63d
OUT: October 10th
DUE: October 24th, 11:59 PM

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 3.4"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only.
- **Late Submission Policy:** Late submissions will not receive full credit. Half credit will be awarded to correct solutions submitted within 48 hours of the original deadline. Otherwise, no credit will be given.
- **Submitting your work:** Non-programming parts of the assignment should be submitted as PDFs using Gradescope unless explicitly stated otherwise. Each derivation/proof should be completed on a separate page. Submissions can be handwritten, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Upon submission, label each question using the template provided.

Problem 1: Probably Approximately Correct (PAC) learning [20] (Hyun-Ah)

In this Problem, we will use following notations:

- X: a set called instance space. A set of all possible instances or examples. (e.g. all points in $\mathbb{R}^2$)
- c: a concept. A subset of the instance space $c \subseteq X$ that exemplify some interesting rule. Our goal is to approximate this concept. $c : X \rightarrow \{0, 1\}$, where $c(x) = 1$ indicates x is a positive example of c.
- $\mathcal{C}$: a concept class. A set of concepts c.
- $\mathcal{D}$: fixed probability distribution over the instance space X. This is the target distribution where the training and test examples are drawn from.
- $\mathcal{S}$: a set of training examples.
- $\mathcal{H}$: a set of concept hypotheses.

Given a set of training samples $\mathcal{S}$, a learning algorithm L chooses a hypothesis h_S from the hypotheses set $\mathcal{H}$ that approximates c.

Then the generalization error of h with respect to the target concept c and distribution $\mathcal{D}$ is given as:

$$R(h) = P_{x\sim\mathcal{D}}[h(x) \neq c(x)] = E_{x\sim\mathcal{D}}[\mathbb{1}_{h(x)\neq c(x)}] \tag{1}$$

The empirical error given m training samples (average error of h on training sample S drawn from $\mathcal{D}$) is given as:

$$\hat{R}_S(h) = P_{x\sim\hat{\mathcal{D}}}[h(x) \neq c(x)] = E_{x\sim\hat{\mathcal{D}}}[\mathbb{1}_{h(x)\neq c(x)}] = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{h(x)\neq c(x)} \tag{2}$$

And we observe that $R(h) = E_{S\sim\mathcal{D}^m}[\hat{R}_S(h)]$.

The formal definition of Provably Approximately Correct (PAC) learning is given as below:

Definition 1. *PAC learnability A concept class $\mathcal{C}$ is PAC learnable if there exists a learning algorithm L such that outputs a hypothesis h_S such that:*

1. *for all $c \in \mathcal{C}$, all distributions $\mathcal{D}$, all $\epsilon > 0$ and all $\delta > 0$, the generalization error of h_S is at most ϵ with probability at least $1 - \delta$, i.e. $P_{S\sim\mathcal{D}^m}[R(h_S) \leq \epsilon] \geq 1 - \delta$*
2. *for samples S of size m that is bounded below by a polynomial in $1/\epsilon$ and $1/\delta$.*

Here, ϵ is an error parameter, i.e. accuracy is $1 - \epsilon$, and δ is the confidence parameter, i.e. confidence is $1 - \delta$.

Now we will see some PAC learnable examples, and work out that they are PAC learnable.

1.1 Rectangle learning [10]

Consider an unknown axis-aligned rectangle R in $\mathbb{R}^2$ space. This R is our target rectangle that we would like to approximate. Assume that you are given information on R by the following procedure: a point p is drawn randomly from distribution $\mathcal{D}$, and you are given the coordinate of this point p and its label indicating whether this point p is included in the rectangle R or not. If the point p is included in R, then the label is 1 and -1 otherwise. Figure 1 shows the R and some points with labels.

Your goal is to use as few examples as possible to pick an approximate to R, i.e. a hypothesis rectangle R′. We will call this "rectangle learning." The error of R′ is computed as the probability that a randomly chosen point fro $\mathcal{D}$ falls in regions $\mathrm{R} \triangle \mathrm{R}'$, where $\mathrm{R} \triangle \mathrm{R}' = (\mathrm{R} - \mathrm{R}') \cup (\mathrm{R}' - \mathrm{R})$.

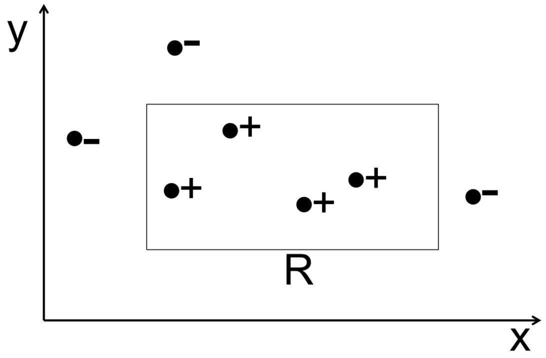


Figure 1: An example of rectangle R and point samples.

1. **[7 pt]** Show that described rectangle learning is PAC learnable. (i.e. for any target rectangle R, and any distribution $\mathcal{D}$, for any small values of $\epsilon > 0$ and $\delta > 0$, for a suitably chosen value of the sample size m, we can assert that with probability at least $1 - \delta$, the tightest-fit rectangle has error at most ϵ with respect to R and D.)
   (hint 1: use union bound: " for any finite or countable set of events, the probability that at least one of the events happens is no greater than the sum of the probabilities of the individual events: $P(\cup_i A_i) \leq \sum_i P(A_i)$ *"*[1]*, and hint 2: use the inequality* $1 - x \leq \exp(-x)$*))*

2. **[1 pt]** What is the lower bound of the sample size m?

3. **[1 pt]** How does m change as we ask for more accuracy (smaller ϵ)?

4. **[1 pt]** How does m change as we ask for narrower confidence interval (smaller δ)?

1.2 Error bounds [10]

1. **[5 pt]** The theorem on the learning bound for finite H for consistent case is given as follows. (consistent case means that hypotheses fits perfectly on the training data: $h_S(x) = c(x), \forall x \in S$, so that we have $\hat{R}_S(h_S) = P_{x \sim \hat{\mathcal{D}}}[h_S(x) \neq c(x)] = 0$.)

   Theorem 1. *let* $\mathcal{H}$ *be a finite set of functions of* X *to* $\{0, 1\}$*, and* L *a learning algorithm that for any target concept* $c \in \mathcal{H}$*, and sample* S *returns a consistent hypothesis* $h_S : \hat{R}_S(h_S) = 0$*. Then for any* $\delta > 0$*, with probability at least* $1 - \delta$*,*

   $$R(h_S) \leq \frac{1}{m}(\log|\mathcal{H}| + \log\frac{1}{\delta}) \tag{3}$$

   Prove above theorem.
   (hint 1: use union bound, and hint 2: use the inequality $1 - x \leq \exp(-x)$*)*

2. **[5 pt]** The theorem on the learning bound for finite $\mathcal{H}$ for inconsistent case is given as follows:

---

[1] https://en.wikipedia.org/wiki/Boole%27s_inequality

Theorem 2. *let H be a finite set, then for any $\delta > 0$, with probability at least $1 - \delta$,*

$$\forall h \in \mathcal{H}, R(h) \leq \hat{R}_S(h) + \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2m}} \tag{4}$$

For the case of inconsistent h, we need following corollary.
Corollary 1. *for any $\epsilon > 0$, and any hypothesis $h : X \rightarrow \{0, 1\}$, the following inequality holds:*

$$P[|R(h) - \hat{R}(h)| \geq \epsilon] \leq 2 \exp(-2m\epsilon^2) \tag{5}$$

Use the corollary on Hoeffding's inequaltiy to prove above theorem.
(note: Hoeffding's inequality provides an upper bound on the probability that the sum of random variables deviates from its expected value. [2]*)*

---

[2] https://en.wikipedia.org/wiki/Hoeffding%27s_inequality

Problem 2: VC Dimension [10] (Brynn)

2.1 Vapnik-Chervonenkis (VC) Dimension [10]

For each one of the following function classes, state what the VC dimension is and explain how you found that number.

1. **[2 pt]** Circles in $\mathbb{R}^2$. An example is labeled positive if it lies within the circle, and negative otherwise. Assume you may shift the circle and change its size.
2. **[2 pt]** Consider the same scenario as in part 1, but now assume the circle is centered around the origin.
3. **[3 pt]**Consider $X \in \mathbb{R}^1$, where we want to learn $c : X \rightarrow \{0, 1\}$. What is the VC dimension of $H = \{Y = sign(sin(w_1 x + w_2))\}$ (Figure 2)

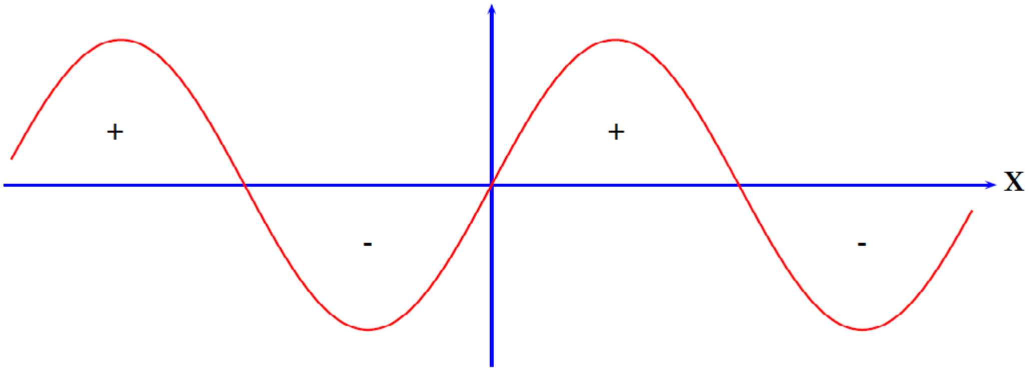


Figure 2: Sine wave classifier

4. **[3 pt]** Prove that the VC dimension of a finite class $|C| < \infty$ is bounded from above: $VCdim(C) \leq log_2|C|$

Problem 3: Perceptron [30] (Siddharth)

3.1 Proving convergence in linear separability [10]

Consider the scenario where we are using linear classifiers (passing through origin) as our learning model ($h(\mathbf{x}) = sign(\mathbf{w^T x})$), and the perceptron algorithm as our learning algorithm. Given a dataset of n samples of d-dimensional vectors ($\mathbf{x_i}$) with class labels (y_i), where the labels can take only 2 values $+1$ or -1, assume that there exists a weight vector $\mathbf{w}^*$ that linearly separates the positive and negative samples, such that:

$$y_i(\mathbf{w}^*)^T \mathbf{x_i} \geq \gamma, \forall i \tag{6}$$

where, $\gamma > 0 (\gamma \in \mathbb{R})$, $\mathbf{x_i} \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$, $\mathbf{w}^* \in \mathbb{R}^d$.

The perceptron algorithm starts with $\mathbf{w^{(0)}} = \mathbf{0}$ (superscript denotes update-step number), and updates the weight vector at the kth step when it encounters a misclassified sample, $\mathbf{w^{(k)}} = \mathbf{w^{(k-1)}} + y_k \mathbf{x_k}$. A sample $(\mathbf{x}, y)$ is misclassified (at a step i) if:

$$y(\mathbf{w^{(i-1)}})^T \mathbf{x} \leq 0 \tag{7}$$

Assume further that for all training input vectors, the L_2 norm is bounded, $||\mathbf{x_i}||_2 \leq V$ ($\forall i = 1..n$).

Show that the number of updates (t) to the weight vector starting from $\mathbf{w^{(0)}} = 0$ in such a scenario is bounded by $\frac{V^2 ||\mathbf{w}^*||_2^2}{\gamma^2}$.

For proving this result:

1. Try to lower bound $\mathbf{w^*}^T \mathbf{w^{(t)}}$, using Equation 6. Specifically show that $\mathbf{w^*}^T \mathbf{w^{(t)}} \geq t\gamma$.
2. Now, upper bound $||\mathbf{w^{(t)}}||^2$ using the update rule (Equation 7), and show that $||\mathbf{w^{(t)}}||_2^2 \leq tV^2$.
3. Now use the above two parts to get the desired result. (Hint: use cosine of $\mathbf{w}^*$, $\mathbf{w^{(t)}}$)

3.2 Implementing perceptron algorithm [7]

Let's consider the linear classification model with a bias term,

$$h(\mathbf{x}) = sign(\mathbf{w^T x} + b) \tag{8}$$

where $sign(v)$ outputs $+1$ if $v \geq 0$, and -1 otherwise.

The perceptron learning algorithm can be expressed as follows (for a given dataset $D = \{\mathbf{x_i}, y_i\}, i = 1..n, \mathbf{x_i} \in \mathbb{R}^d, y_i \in \{+1, -1\}$):

```
Let $\mathbf{w} = \mathbf{0}$, $i = 0$, $numIter = 100$
while !(all samples correctly classfied) AND $i < numIter$ do
    for $s = 1, 2, ..n$ do
        if $y_s(\mathbf{w^T x_s} + b) \leq 0$ then
            $\mathbf{w} = \mathbf{w} + y_s \mathbf{x_s}$;
            $b = b + y_s$ ;
        end
    end
    $i = i + 1$ ;
end
return $\mathbf{w}, b, i$
```

Algorithm 1: Perceptron algorithm on training samples

This algorithm can be used to obtain $\mathbf{w}$ and b, which can then be used to predict labels of the test samples using equation 8. Implement this algorithm for the task of classification on the dataset given *here* (filenames with ‘parta’ prefix). Report your accuracy on the test set, and the number of iterations it took the algorithm to converge on the training set.

3.3 Implementing kernel perceptron algorithm [13]

Let’s consider a kernel-defined feature space $\boldsymbol{\phi}(\mathbf{x})$ ($\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^E, E >> d$), and the linear classification model in that space. More precisely, the model is defined as:

$$g(\mathbf{x}) = sign(\mathbf{w^T}\boldsymbol{\phi}(\mathbf{x})) \tag{9}$$

(Note, that we are not considering the bias term in the new feature space). So in order to apply the perceptron learning algorithm, the update rule will have to be modified. The update rule then becomes $\mathbf{w^{(t)}} = \mathbf{w^{(t)}} + y_i\boldsymbol{\phi}(\mathbf{x_i})$ for the misclassified example $(\mathbf{x_i}, y_i)$. But this generally turns out to be computationally expensive, so we consider the dual version of the problem. Assuming n dual variables α_i corresponding to each training input vector (which are initialized to 0), the dual update rule is $\alpha_i = \alpha_i + 1$ (for the above ith misclassified sample).

The weight vector (at iteration t) can be expressed as:

$$\mathbf{w^{(t)}} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x_i}) \tag{10}$$

Subsequently, the function g (from equation 9), for an input $\mathbf{x}$, can be written as:

$$g(\mathbf{x}) = sign(\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x_i})^T \boldsymbol{\phi}(\mathbf{x})) \tag{11}$$

(Note, $\mathbf{x_i}$ is the ith point in the dataset).

Now, we can see that we can use kernel inner products instead of using $\boldsymbol{\phi}(\mathbf{x})$ directly. Moreover, we don’t need an explicit form of the weight vector to compute $g(\mathbf{x})$, it can written as:

$$g(\mathbf{x}) = sign(\sum_{i=1}^{n} \alpha_i y_i \kappa(\mathbf{x_i}, \mathbf{x})) \tag{12}$$

In conclusion, we only need values of α_i for predicting labels for test samples.

The kernel perceptron algorithm can be written as follows (note: $\boldsymbol{\alpha} \in \mathbb{R}^n$, α_i is ith component (hence a scalar)):

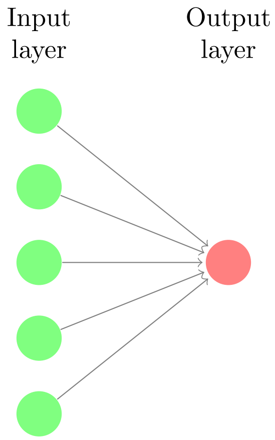
```
Let $\boldsymbol{\alpha} = \mathbf{0}$, $i = 0$, $numIter = 100$
while !(all samples correctly classified) AND $i < numIter$ do
    for $s = 1, 2, ..n$ do
        if $y_s \times \sum_{j=1}^{n}(\alpha_j y_j \kappa(\mathbf{x_j}, \mathbf{x_s})) \leq 0$ then
            $\alpha_s = \alpha_s + 1$ ;
        end
    end
    $i = i + 1$ ;
end
return $\alpha$
```

Algorithm 2: Kernel perceptron algorithm on training samples

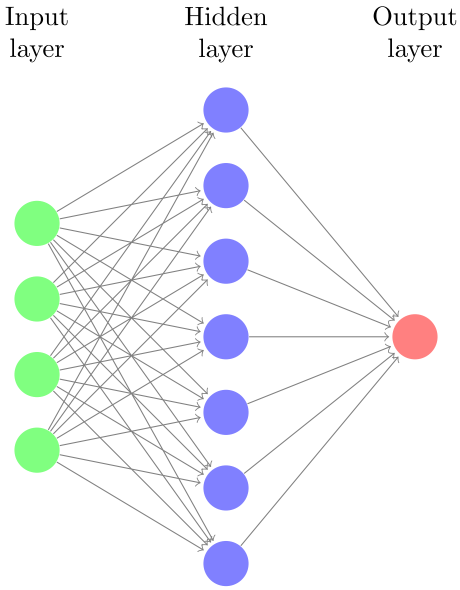
Implement the kernel perceptron algorithm on this *dataset* (filenames with 'partb' prefix) using a polynomial kernel of degree 3 (which has the form $\kappa(\mathbf{x_i}, \mathbf{x_j}) = (1 + \mathbf{x_i^T x_j})^c$ for degree c). Use the training dataset to obtain suitable dual variables (the vector $\boldsymbol{\alpha}$), and use equation 12 to predict labels for the test samples. Use $numIter = 10$ for the kernel perceptron algorithm, and compare its accuracy against that obtained by the perceptron learning algorithm on this dataset (with $numIter = 100$ for perceptron learning algorithm).

Problem 4: Neural networks [30] (Brynn)

4.1 Network Architecture [16]



1. **[1 pt]** What is the most common name of the neural network shown in the figure above, when sigmoid activation functions are being used?
2. **[1 pt]** What does the decision boundary of the neural network shown in the figure above look like when sigmoid activation functions are being used?



3. **[1 pt]** Now consider the model shown in the figure above, instead of the previous one. Why might this be a better model?
4. **[1 pt]** And why could it be worse?
5. **[8 pt]** Assume the following functional form for the network shown in the above figure:

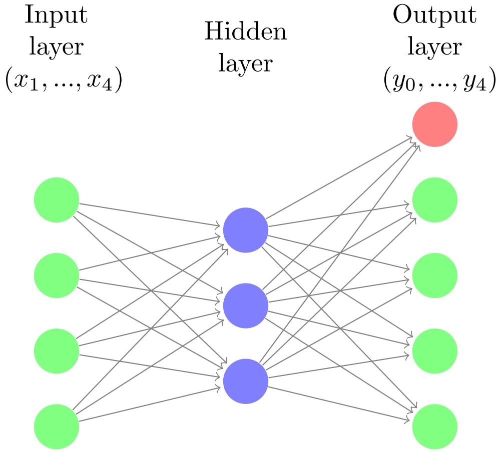
$$h(x) = \sigma(W_h x + b_h),$$
$$o(x) = \sigma(W_o h(x) + b_o),$$

where h corresponds to the vector of hidden node values, o corresponds to the output node value, $\sigma(\cdot)$ is the sigmoid function and $\{W_h, b_h, W_o, b_o\}$ are the parameters of the network. Assume that we use

the squared error loss function and that we have available training data $\{x_i, y_i\}_{i=1}^n$. Derive the gradient of the total loss with respect to parameters b_o and b_h.
Hint: The total loss is the sum of loss for each data point where loss function for each data point is given by the equation:

$$l(x_i, y_i) = \|o(x_i) - y_i\|_2^2$$

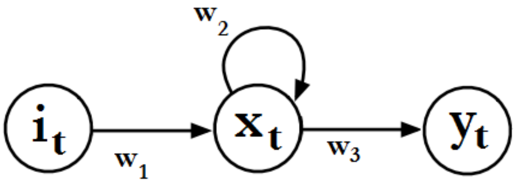
6. **[1 pt]** Which quantities in the above derivation correspond to the quantities that we *back-propagate* in the back-propagation algorithm?
7. **[1 pt]** Why do we use back-propagation and not, let's say, recursion using a forward pass on the neural network, to compute the gradients?



8. **[2 pt]** Suppose we wish to learn the function $f : \langle x_1, x_2, x_3, x_4 \rangle \to y_0$, where y_0 is the node depicted in red. Consider this network, where we train it to produce outputs $y_1 = x_1, y_2 = x_2, y_3 = x_3, y_4 = x_4$, as well as $y_0 = f(x_1...x_4)$. What is this network doing, and why might it be a useful way to train $f : \langle x_1, x_2, x_3, x_4 \rangle \to y_0$?

4.2 Recurrent Neural Networks [14]

Consider the following recurrent neural network (RNN):



$$y_t = w_3 x_t,$$

$$x_t = \sigma(w_2 x_{t-1} + w_1 i_t)$$

where i_t denotes some input, x_t denotes some hidden unit, and y_t denotes a target output.

Suppose you wish to train this network using gradient descent to fit the input/output time series $\langle\langle i_1, y_1\rangle...\langle i_T, y_T\rangle\rangle$. Derive the gradient descent rule for training this network. That is, calculate the gradient including each network parameter, and give the training algorithm. Derive your algorithm to minimize the sum of squared errors $\sum_{t=1}^{T}(\hat{y}_t - y_t)^2$. Assume when calculating y_1, that $x_0 = 1$.