CS446: Machine Learning **Fall 2016**

Problem Set 4

Handed Out: October 11^{th}*,* 2015 *Due: October* 20^{th}*,* 2016

- Feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You should, however, write down your solution yourself. Please try to keep the solution brief and clear.
- Please use Piazza first if you have questions about the homework. Also feel free to send us e-mails and come to office hours.
- Please, no handwritten solutions. You will submit your solution manuscript as a single pdf file.
- Please present your algorithms in both pseudocode and English. That is, give a precise formulation of your algorithm as pseudocode and *also* explain in one or two concise paragraphs what your algorithm does. Be aware that pseudocode is much simpler and more abstract than real code.
- The homework is due at 11:59 PM on the due date. We will be using Compass for collecting the homework assignments. Please submit your solution manuscript as a pdf file via Compass (`http://compass2g.illinois.edu`). Please do NOT hand in a hard copy of your write-up. Contact the TAs if you are having technical difficulties in submitting the assignment.
- You can only use 24 late submission credit hours for this problem set. We will release the solution 24 hours after it is due so that you can have time to go over it before the mid-term on Oct. 25th.

1. **[VC Dimension - 30 points]**

   (a) **[15 points]** Assume that all examples are points in a two-dimensional space, i.e. $\mathbf{x} = \langle x_1, x_2 \rangle \in \mathbb{R}^2$. Consider the concept space of triangles in the plane. Hence, a concept $h \in \mathcal{H}$ is specified by three vertices $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \langle x_3, y_3 \rangle$ (that are not on the same line). An example $\mathbf{x} \in \mathbb{R}^2$ is labeled as positive by h if and only if $\mathbf{x}$ lies inside or on the border of the triangle. Give the VC dimension of $\mathcal{H}$ and prove that your answer is correct. [Hint: When proving the upper bound of the VC dimension, use the concept of *convex hull* to distinguish among different situations. Provide illustrative pictures if necessary.]

   (b) **[15 points]** Consider the concept space of axis aligned rectangles in $\mathbb{R}^d$ where $d \in \mathbb{Z}_+$. Here, a concept $h \in \mathcal{H}$ is represented by $2d$ parameters $a_i < b_i$ for $i \in [1, \ldots, d]$. An example $\mathbf{x} \in \mathbb{R}^d$ is labeled as positive by h if and only if

   $$\forall i \in [1, \ldots, d] : a_i \leq x_i \leq b_i$$

   Give the VC dimension of $\mathcal{H}$ and prove that your answer is correct. [Hint: try proving the VC dimension for $d = 2$ and then generalizing for larger dimensions].

   Grading note: You will not get any points without proper justification of your answer.

2. **[Kernels - 30 points]**

   (a) **[5 points]** Write down the dual representation of the Perceptron algorithm.

(b) **[5 points]** Given two examples $\vec{\mathbf{x}} \in \mathbb{R}^2$ and $\vec{\mathbf{z}} \in \mathbb{R}^2$, let

$$K(\vec{\mathbf{x}}, \vec{\mathbf{z}}) = (\vec{\mathbf{x}}^T \vec{\mathbf{z}})^3 + 400(\vec{\mathbf{x}}^T \vec{\mathbf{z}})^2 + 100\vec{\mathbf{x}}^T \vec{\mathbf{z}}.$$

Prove that this is a valid kernel function.

For the rest of this problem, we wish to learn a Boolean function represented as a **monotone** DNF (DNF without negated variables) using kernel Perceptron. For this problem, assume that the size of each term in the DNF is bounded by k, i.e., the number of literals in each term of the DNF is between 1 to k. In order to complete this task, we will first define a kernel that maps an example $\mathbf{x} \in \{0,1\}^n$ into a new space of monotone conjunctions of **up to** k different variables from the n-dimensional space. Then, we will use the kernel Perceptron to perform our learning task.

(c) **[5 points]** Define a kernel $K(\mathbf{x}, \mathbf{z}) = \sum_{c \in C} c(\mathbf{x})c(\mathbf{z})$, where C is a family of monotone conjunctions containing **up to** k different variables, and $c(\mathbf{x}), c(\mathbf{z}) \in \{0,1\}$ is the value of c when evaluated on example $\mathbf{x}$ and $\mathbf{z}$ separately. Show that $K(\mathbf{x}, \mathbf{z})$ can be computed in time that is linear in n. [Hint: It would be useful to think about the case where each term of the DNF is *exactly* of k literals and then generalize.]

(d) **[15 points]** State a bound on the number of mistakes the kernel Perceptron algorithm will make on a sample of examples consistent with a function in this class. [Hint: In class, we proved a theorem by Novikoff; use this bound and estimate the two constants in it for this specific case. When doing it, observe that you are no longer learning in the original n dimensional space, but rather in a blown-up space that has a different dimensionality.]

3. **[Neural Networks - 40 points]** For this problem, you will construct a neural network to solve a non-linear classification task. Each node in your neural network will use the following activation function:

$$f(x) = \max(0, x)$$

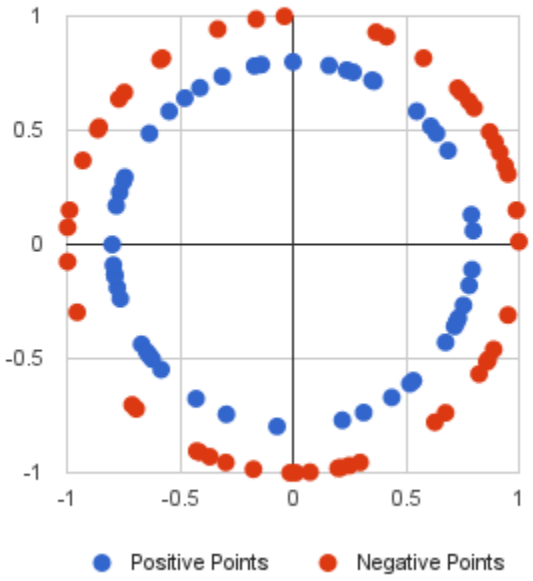
This function is called the **rectifier**; a neural network node using the rectifier for its activation function is often called a **ReLU**, which stands for rectified linear unit. Additionally, you will use the squared error loss function, defined as the following:

$$Err(x_i, w) = \frac{1}{2} \sum_{k \in K} (t_k - o_k)^2$$

Where K is the set of nodes in the output layer, t_d is the correct value for output node d, and o_d is the output of node d.

(a) **[10 points]** Derive the backpropagation weight update rules for this activation function and loss. Note that there are two different kinds of updates you need to find: one representing weights for the nodes in the output layer, and one representing weights for the nodes in the hidden layers.

(b) **[30 points]** For this problem, we will experiment with training a neural network for learning two different functions. First, we have generated a dataset consisting of two concentric circles; the points on the inner circle are labeled as positive, and the points on the outer circle are labeled as negative. Here is a small sample of the dataset:



The second dataset consists of a subset of the mnist digits dataset [1]. This dataset consists of images of hand-drawn (single) digits; for this assignment, you will be receiving a subset consisting of 3s and 8s. Specifically, we are asking you to do the following:

i. **[10 points]** For your convenience, we have already implemented most of the backpropagation algorithm for you. However, we have left out a few important computations. To complete this, implement the following functions, found in `NN_functions.py`:

- `squared_loss_gradient(output, label)`: this should return the gradient of the squared loss with respect to each coordinate of the output.
- `relu_derivative(z)`: this should return the derivative of the rectifier function $f(z) = \max(0, z)$.

ii. **[10 points]** Using the provided neural network code (see description below for more details), run a 5-fold cross validation parameter tuning experiment to find the top-performing parameter setting for each dataset. See below for a description of the parameters being tuned and the specific values you should try. Specifically, for each parameter setting, complete the following steps:

[1] http://yann.lecun.com/exdb/mnist/

- Split the training data into 5 portions of equal size
- For each split: train on the rest of the training data and test on the held-out split.
- Record the average accuracy over the splits.

For both data sets, report the average accuracy for each parameter setting, and make a note of which parameter setting performed the best.

iii. **[10 points]** We want you to compare the performance of the neural network with a simple Linear Classifier. To that end, we have provided you with an implementation of the Perceptron algorithm to train linear separators for each dataset. For this task, you will have to generate a learning curve (accuracy vs. number of iterations) during training for both Perceptron and the neural network using the best parameter settings found in the previous step. For each data set, plot these learning curves on the same graph; thus, your answer will include two graphs, one for each dataset. You have been provided with functions that keep track of these values for your convenience (see below). After training your models on a given data set, test each of them using the corresponding test data. Record the accuracy of both models on the test data for each of the two data sets and comment on their performances relative to each other.

Parameter Tuning: One crucial aspect of training Machine Learning models is to tune the available free parameters so that you can achieve the best performance from your learning algorithm. The parameters depend highly on the dataset and also on the complexity of the task. When training neural networks, there are some critical decisions to make regarding the structure of the network and the behaviour of its individual units. In this section we describe some parameters that you will tweak while training your classifier:

- Batch Size for training : This is the number of examples that are processed at the same time. Use the following values: $[10, 50, 100]$
- Activation Function : The (nonlinear) function applied at the end of computation for each node. Use the ReLU and tanh activation functions.
- Learning rate : The learning rate for gradient descent. Use the following values: $[0.1, 0.01]$.
- Number of units in each hidden layer : The number of nodes contained within each hidden layer. Use the following values: $[10, 50]$

Experiment Code: The code consists of the following files:

- `data_loader.py`: Contains the function `load_data()`, which loads the dataset from the files and initializes it in the appropriate way
- `NN.py`: Contains the implementation of the neural network training/testing procedures. Take note of the function `create_NN(batch_size, learning_rate, activation_function, hidden_layer_width)` - this takes in the specified parameters and correctly returns an instance of the `NN` class, which will simplify the process of initializing the neural network for parameter tuning purposes.

- `NN_functions.py`: Contains functions used during neural network training/testing. **This file contains the two functions mentioned earlier that need to be completed.**
- `perceptron.py`: Contains a perceptron implementation.
- `sample_run.py`: Contains a sample showing how to use the provided code. **This is the best resource to learn how to run the provided code**.

Both the `NN` and `Perceptron` classes contain the following functions:

- `train(self, training_data)`: Trains the classifier represented by the object. Returns the final accuracy on the training data.
- `train_with_learning_curve(self, training_data)`: Trains the classifier represented by the object while keeping track of performance at each iteration. Returns a list of tuples (i, acc_i) where acc_i is the accuracy of the classifier after training for i iterations.

We have included a README providing more information about running the code.

What to Submit

- A pdf file which contains answers to each question.
- Your source code. This should include your implementation of the missing neural network functions and the code that runs your experiments. You **must** include a README, documenting how someone should run your code.
- Please upload the above three files on Compass. (`http://compass2g.illinois.edu`)