CS446: Machine Learning **Spring 2017**

Problem Set 5

Handed Out: March 30^{th}, 2017 *Due: April* 11^{th}, 2017

- 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.

1. [**Neural Networks - 50 points**] For this problem, you will construct a single hidden layer neural network to solve a non-linear classification task. Each node in your neural network will use the following activation function:

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

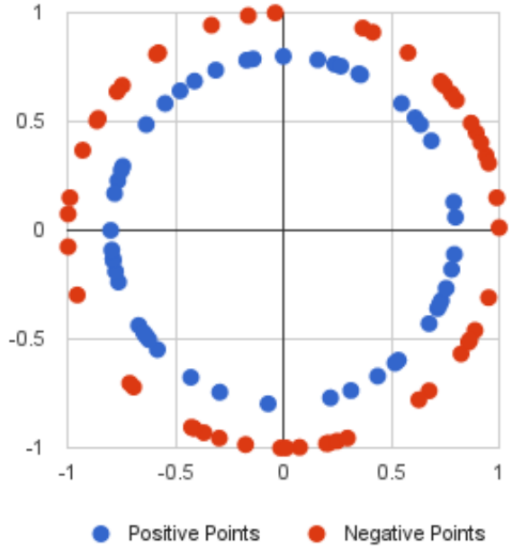
This function is called the **rectifier**; a neural network node using the rectifier for its activation function is often called a **ReLU**, which stands for rectified linear unit. Additionally, you will use the squared error loss function, defined as the following:

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

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 of connections between the output layer and the hidden layer, and one representing weights for the connections between the hidden layer and the input layer.

(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:

   - 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.

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

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.

2. [**Multi-class classification - 30 points**]

Consider a multi-class classification problem with k class labels $\{1, 2, \dots k\}$. Assume that we are given m examples, labeled with one of the k class labels. Assume, for simplicity, that we have m/k examples of each type.

Assume that you have a learning algorithm L that can be used to learn Boolean functions. (E.g., think about L as the Perceptron algorithm). We would like to explore several ways to develop learning algorithms for the multi-class classification problem.

There are two schemes to use the algorithm L on the given data set, and produce a multi-class classification:

- **One vs. All:** For every label $i \in [1, k]$, a classifier is learned over the following data set: the examples labeled with the label i are considered "positive", and examples labeled with any other class $j \in [1, k], j \neq i$ are considered "negative".
- **All vs. All:** For every pair of labels $\langle i, j \rangle$, a classifier is learned over the following data set: the examples labeled with one class $i \in [1, k]$ are considered "positive", and those labeled with the other class $j \in [1, k], j \neq i$ are considered "negative".

(a) [**5 points**] For each of these two schemes, answer the following:

i. How many classifiers do you learn?

ii. How many examples do you use to learn each classifier within the scheme?

iii. How will you decide the final class label (from {1, 2, . . . , k}) for each example?

iv. What is the computational complexity of the training process?

(b) [**5 points**] Based on your analysis above of two schemes individually, which scheme would you prefer? Justify.

(c) [**5 points**] You could also use a KernelPerceptron for a two-class classification. We could also use the algorithm to learn a multi-class classification. Does using a KernelPerceptron change your analysis above? Specifically, what is the computational complexity of using a KernelPerceptron and which scheme would you prefer when using a KernelPerceptron?

(d) [**5 points**] We are given a magical black-box binary classification algorithm (we dont know how it works, but it just does!) which has a learning time complexity of O(dn^2), where n is the total number of training examples supplied (positive+negative) and d is the dimensionality of each example. What are the overall training time complexities of the all-vs-all and the one-vs-all paradigms, respectively, and which training paradigm is most efficient?

(e) [**5 points**] We are now given another magical black-box binary classification algorithm (wow!) which has a learning time complexity of $\mathrm{O}(d^2 n)$, where n is the total number of training examples supplied (positive+negative) and d is the dimensionality of each example. What are the overall training time complexities of the all-vs-all and the one-vs-all paradigms, respectively, and which training paradigm is most efficient, when using this new classifier?

(f) [**5 points**] Suppose we have learnt an all-vs-all multi-class classifier and now want to proceed to predicting labels on unseen examples.

We have learnt a simple linear classifier with a weight vector of dimensionality d for each of the $m(m-1)/2$ classes ($w_i^T x = 0$ is the simple linear classifier hyperplane for each $i = [1, \cdots, m(m-1)/2]$)

We have two evaluation strategies to choose from. For each example, we can:

- **Counting**: Do all predictions then do a majority vote to decide class label
- **Knockout**: Compare two classes at a time, if one loses, never consider it again. Repeat till only one class remains.

What are the overall evaluation time complexities per example for Counting and Knockout, respectively?

3. [**Probability Review (20 points)**]

(a) There are two towns A and B, where all families follow the following scheme for family planning:

- Town A: Each family has just one child – either a boy or a girl.
- Town B: Each family has as many children as it wants, until a boy is born, and then it does not have any more children.

Assume that the boy to girl ratio is 1:1 for both towns A and B (number of boys equals number of girls), and the probability of having a boy child is 0.5, the same as that of having a girl child. Answer the following questions:

i. What is the expected number of children in a family in towns A and B?

ii. What is the boy to girl ratio at the end of one generation in towns A and B?

(b) i. For events A and B, prove

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

ii. For events A, B, and C, rewrite $P(A, B, C)$ as a product of several conditional probabilities and one unconditional probability involving a single event. Your conditional probabilities can use only one event on the left side of the conditioning bar. For example, $P(A|C)$ and $P(A)$ would be okay, but $P(A, B|C)$ is not.

(c) Let A be any event, and let X be a random variable defined by

$$X = \begin{cases} 1 & \text{if event } A \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

X is sometimes called the *indicator* random variable for the event A. Show that $\mathbb{E}[X] = P(A)$, where $\mathbb{E}[X]$ denotes the expected value of X.

(d) Let X, Y, and Z be random variables taking values in $\{0, 1\}$. The following table lists the probability of each possible assignment of 0 and 1 to the variables X, Y, and Z:

	$Z = 0$		$Z = 1$	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
$Y = 0$	1/15	1/15	4/15	2/15
$Y = 1$	1/10	1/10	8/45	4/45

For example, $P(X = 0, Y = 1, Z = 0) = 1/10$ and $P(X = 1, Y = 1, Z = 1) = 4/45$.

i. Is X independent of Y ? Why or why not?

ii. Is X conditionally independent of Y given Z? Why or why not?

iii. Calculate $P(X = 0 | X + Y > 0)$.

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`)